1997-03

A study of video teleconferencing traffic on a TCPIP network

Carvey, Harlan A.

Monterey, California. Naval Postgraduate School

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A STUDY OF VIDEO TELECONFERENCING TRAFFIC ON A TCP/IP NETWORK

by

Harlan A. Carvey

March 1997

Thesis Advisor Murali Tummala

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE TITLE OF THESIS A Study of Video Teleconferencing Traffic on a TCP/IP Network	5. FUNDING NUMBERS
6. AUTHOR(S) Harlan A. Carvey	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

In this thesis the nature of variable bit rate (VBR) traffic, as generated by a video teleconferencing application, in an Ethernet environment is studied. Analysis of the data retrieved from a testbed using metrics such as the rescaled adjusted range statistic, variance-time curve, and index of dispersion for counts illustrate the self-similar nature of traffic generated by a video teleconferencing application. This information is useful in formulating accurate models to support the various classes of traffic that will dominate the broadband ISDN (B-ISDN or ATM) infrastructure and in developing adequate access control mechanisms for those classes of traffic. The future of wide-area networking will see Ethernet LANs populating the access points of ATM WANs, thus making use of the ATM transport mechanism for wide-area communications. This thesis reports on work pertaining only to the traffic offered by the Ethernet LAN.

Java and the Simple Network Management Protocol (SNMP) provide the means with which to construct tools for gathering and simulating VTC traffic. Java applets were written to measure and simulate VTC traffic.

14. SUBJECT TERMS Video Teleconferencing, variable bit rate, VBR, SNMP, Java	15. NUMBER OF PAGES 86
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

A STUDY OF VIDEO TELECONFERENCING TRAFFIC ON A TCP/IP NETWORK

Harlan A. Carvey
Captain, United States Marine Corps
B.S., Virginia Military Institute, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

March 1997

ABSTRACT

In this thesis the nature of variable bit rate (VBR) traffic, as generated by a video teleconferencing (VTC) application, in an Ethernet environment is studied. Graphical analysis of the data retrieved from a testbed using metrics such as the rescaled adjusted range statistic, variance-time curve, and index of dispersion for counts illustrates the self-similar nature of traffic generated by a video teleconferencing application. Additionally, similar analysis is conducted using simulated traffic generated via statistical models. This information is useful in formulating accurate models to support the various classes of traffic that will dominate the broadband ISDN (B-ISDN or ATM) infrastructure and in developing adequate access control mechanisms for those classes of traffic. The future of wide-area networking will see Ethernet LANs populating the access points of ATM WANs, thus making use of the ATM transport mechanism for wide-area communications. This thesis reports on work pertaining only to the traffic offered by the Ethernet LAN.

Java and the Simple Network Management Protocol (SNMP) provide the means with which to construct tools for gathering and simulating VTC traffic. Java applets were written to measure and simulate VTC traffic.

TABLE OF CONTENTS

I. INTRODUCTION 1
II. VIDEO TELECONFERENCING 3
A. VIDEO TELECONFERENCING SYSTEM 3
B. VIDEO TELECONFERENCING 3
C. STANDARDS AND COMMERCIAL SYSTEMS 7
III. TESTBED NETWORK 9
A. TESTBED ENVIRONMENT 9
B. VIDEO TELECONFERENCING SOFTWARE 11
C. VIDEO/AUDIO CAPTURE CARD 11
D. SIMPLE NETWORK MANAGEMENT PROTOCOL 12
E. JAVA 13
IV. TRAFFIC MODELING 17
A. SELF-SIMILAR NATURE OF TRAFFIC 17
B. MINISOURCE 19
C. SIMULATION 21
V. RESULTS 25
A. DATA COLLECTION 25
B. RESULTS 26
1. R/S Statistic 29
2. Variance-Time Curve 31
3. Index of Dispersion for Counts (IDC) 33
C. RESULTS FOR VIDEO/AUDIO TRAFFIC 35
VI. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK 43
A. CONCLUSIONS 43
B. RECOMMENDATIONS FOR FUTURE WORK 44
APPENDIX A: JAVA-SNMP APPLET SOURCE CODE 45
A. MIBCRUSHER.JAVA 45
B. POLLTHREAD.JAVA 49
C. SNMPGET.JAVA 52
D. MIBCRUSHER.HTML 54
APPENDIX B: JAVA SIMULATION APPLET SOURCE CODE 55
A. SIMTEST.JAVA 55
B. SIMTEST2.JAVA 62
APPENDIX C: MATLAB SOURCE CODE 71
A. TRAFFIC.M 71
B. RS.M 72
C. VAR.M 75
D. IDCM.M 78
LIST OF REFERENCES 83
INITIAL DISTRIBUTION LIST 85

LIST OF FIGURES

Figure 2.1: VTC Platform 4
Figure 2.2: Point-to-Point Communication Model 5
Figure 2.3: BroadCast Communications Model 5
Figure 2.4: BroadCast Communications Model (with Reflector) 6
Figure 2.5: Multicast Communication Model 6
Figure 3.1: Testbed System Configuration 9
Figure 3.2: Testbed Topology 10
Figure 3.3: Layered Model 10
Figure 3.4: VTC System Diagram 11
Figure 3.5: SNMP Communication Model 12
Figure 3.6: Layered Model with SNMP 13
Figure 3.7: Java-SNMP Applet Flow Diagram 14
Figure 3.8: Graphical User Interface for Java-SNMP Applet 14
Figure 4.1: Two-State On-Off Model 17
Figure 4.2: Markov Chain Representation of VBR Source 18
Figure 4.3: Video Source Simulation Applet Flow Diagram 19
Figure 4.4: Graphical User Interface for VBR Simulation Applet 20
Figure 4.5: Audio/Video Simulation Applet Flow Diagram 21
Figure 5.1: Input Traffic into Computer Interface, Video Only 27
Figure 5.2: Simulated Traffic 28
Figure 5.3: Plot of R/S Statistic, Video-Only Traffic (Measured) 30
Figure 5.4: Plot of R/S Statistic, Video-Only Traffic (Simulated) 31
Figure 5.5: Variance-Time Curve, Video-Only Traffic (Measured) 32
Figure 5.6: Variance-Time Curve, Video-Only Traffic (Simulated) 33
Figure 5.7: IDC Curves for Video-Only Measured and Simulated Traffic 34
Figure 5.8: Input Traffic at Computer Interface, Video/Audio 35
Figure 5.9: Simulated Video/Audio Traffic 36
Figure 5.10: Plot of R/S Statistic, Video/Audio Traffic (Measured) 37
Figure 5.11: Plot of R/S Statistic, Video/Audio Traffic (Simulated) 38
Figure 5.12: Variance-Time Curve, Video/Audio Traffic (Measured) 39
Figure 5.13: Variance-Time Curve, Video/Audio Traffic (Simulated) 34
Figure 5.14: IDC Curves for Video/Audio Measured and Simulated Traffic 35

ACKNOWLEDGEMENTS

I would first like to thank my wife, Hannah, for her thoughtfulness and understanding throughout our entire tour here at the Naval Postgraduate School. She kept me on track, and often let me know when spending too much time with the computers in the testbed lab. Thank you Hannah.

A particular expression of gratitude goes to Professor Murali Tummala. When the resident networking expert in the ECE curriculum was no longer available, Professor Tummala had the interest and the funds available for me to gain valuable hands-on experience. He provided me with an empty room, a general plan of what he expected, and he cut me loose to design and implement a computer network, to include several computers and two routers. Without his interest and support, this thesis would not have been possible.

I. INTRODUCTION

As digital communication networks expand, they are becoming a more popular and highly viable means of communicating in real-time over vast distances. The networks themselves have evolved from an interconnection of a few PCs in an office on a low bit rate (i.e., 4 Mbps) local area network (LAN) to several hundred or even a thousand machines of various makes and models connected over different media from coast to coast and around the world. As the networks expand, so do the demands placed on them, especially as they move from providing non-real-time data to including real-time, interactive communications as well.

In order to best predict how the overall wide area network (WAN) will react to additions of LANs and make the most efficient use of the resources, some means of understanding expected traffic patterns is necessary. There are many tools available that allow network engineers to model and simulate the network, in order to analyze how a small change will affect the overall infrastructure. However, in order to do this, accurate models of the types of traffic that can be expected to be encountered are needed. Data traffic, such as file transfers and electronic mail, is relatively easy to model [Subramanian 1995], but as new types of traffic are introduced, new models are required. More importantly, traffic observed on networks is no longer simply non-real-time data traffic. Early networks were designed to accommodate data traffic, which is predominantly bursty in nature and requires reliable service from the network, but can tolerate some delay. Interactive multimedia traffic, consisting of audio and video data streams, has now come to dominate these networks. The multimedia traffic is real-time and stream-oriented in nature, and must be delivered with a bounded delay though some loss of data may be tolerated. With recent improvements in network and computing technology this traffic has become more predominant. Studies have been done in order to analyze and accurately model this new traffic as it appears on Ethernet networks [Leland 1994].

This thesis examines a traffic model specific to video teleconferencing (VTC). The intent is to isolate this traffic on an ethernet testbed and derive the necessary information to validate a statistical model for the variable bit rate (VBR) traffic generated by a VTC system in an ethernet environment as well as the traffic generated by a simulation of the minisource model. As VBR traffic is expected to dominate networks in the future, an understanding of the nature of the traffic to be offered at the access points

of the broadband network is extremely important when developing adequate admission, access and flow control mechanisms.

The objective of this thesis is to determine if a single type of traffic, specifically VBR traffic generated by a desktop VTC system, adheres to a known and previously predicted model of self-similarity. No attempts are made to determine the degree of self-similarity of the traffic. The goal is to verify that the traffic exhibits the characteristics for self-similarity as given by [Fowler 1991], [Leland 1994], and [Subramanian 1995].

The purpose of this thesis is to present the results of gathering data from the testbed network. Chapter II provides background information on video teleconferencing in general. Chapter III provides specific information on the testbed network and tools used in this thesis. Chapter IV reviews information pertinent to the formulation of a statistical traffic model, specifically of a "self-similar" or "fractal" nature. Information on determining whether a data sequence is "self-similar" is of particular interest while determining the degree of "self-similarity" is not. Chapter V presents the tests conducted in this thesis and the data derived from them. Chapter VI contains conclusions and recommendations for future work.

II. VIDEO TELECONFERENCING

This chapter presents information pertaining to the importance of VTC and background on how VTC is conducted. The chapter begins by discussing some basics of VTC and then reviews models of communication, and discusses some commercial VTC systems.

A. VIDEO TELECONFERENCING SYSTEM

With the advent in recent years of faster, more powerful, and significantly cheaper computer components, communicating over networks has become increasingly popular. Due to the tremendous popularity of the Internet, the computer networks have expanded to include geographically-dispersed WANs outside of the local intranets and LANs. Upgrades in software and hardware have made multimedia communications increasingly popular, and many applications have been developed to take advantage of the TCP/IP (Transport Control Protocol/Internet Protocol) protocol suite [Stevens 1994]. Video teleconferencing has also gained popularity, both as a user application and as an area of research. Many video teleconferencing software applications, both commercial and shareware/freeware, are readily available and easy to use. With minor upgrades of sound and video cards (in order to send and receive audio and video data streams, respectively) current and legacy systems can be easily deployed into a fully-functional video teleconferencing platform (see Figure 2.1).

B. VIDEO TELECONFERENCING

Video teleconferencing involves sharing multimedia data between two or more users in distinct, often remote locations. The multimedia data consists of predominantly, but is not limited to, video and audio data streams. The data may also include text-mode "chat" systems or networked collaborative "whiteboard" applications in which users share a common screen or application. In the past, VTC was a method of communication thought to be relegated to large corporations and research institutions due to the high expense of the components of such a system.

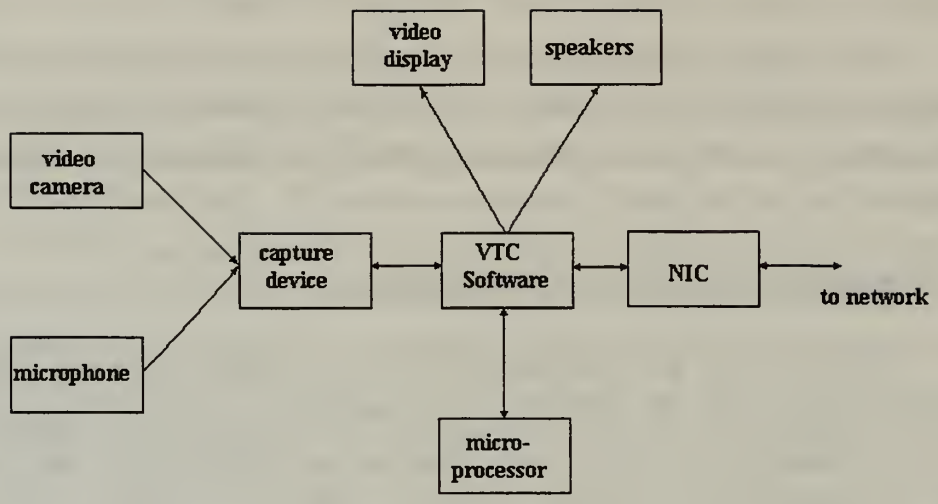


Figure 2.1: VTC Platform

VTC is currently used, or can be used, in almost any situation that requires two or more people to communicate over geographically-dispersed locations. VTC is increasingly being used in such areas as distance learning [Harju 1994], [Vetter 1995] and telemedicine [Perloff 1993]. This increased use has also led to increased research in the disciplines that support teleconferencing: video capture and compression techniques, audio compression and transmission in packet-switched environments, and networking technologies. With advances in these areas and in computer technology in recent years, VTC as an application has moved from the corporate and research arenas to the individual user desktop. Processing power of microprocessors has increased to the point where specially-developed workstations are no longer required to conduct a VTC session, and modem and network speeds have increased enough so that users with dial-up or Ethernet connections, respectively, to the Internet have sufficient capabilities to conduct a VTC session.

VTC is accomplished in one of several communications models, such as point-to-point, one-to-many or broadcast, and multicast [Cobbley 1993]. Point-to-point teleconferencing on a LAN or over the Internet takes place when two users establish a connection between their systems through the use of their respective IP addresses, making use of the connection-oriented TCP, while other systems establish a connection via a dedicated Integrated Services Digital Network (ISDN) line, as in Figure 2.2. Once this

connection is established, the two users can share applications and documents, transmit video and audio between their locations, and use collaborative applications.

Figure 2.2: Point-to-Point Communication Model

Broadcasting involves three or more users in a connection-oriented, video teleconferencing environment, with each user addressing multiple users. To be interactive, this model would require that every user is connected to every other user. Intuitively, this model tends to quickly lead to congestion on networks or network links that cannot provide the appropriate bandwidth (see Figure 2.3). However, this model may be adapted such that all users are connected through a central distribution or "reflector" site. In this model, all users send and receive multimedia traffic to and from all other users via reflector site "R", again using TCP (see Figure 2.4).

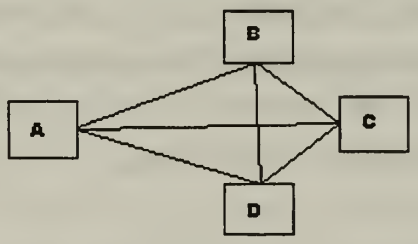


Figure 2.3: Broadcast Communications Model

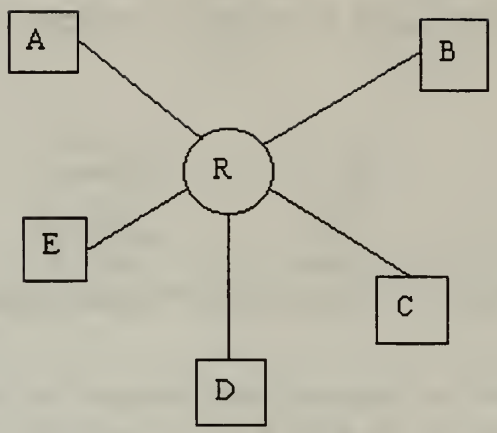


Figure 2.4: Broadcast Communications Model (with Reflector)

An alternative to broadcasting is multicasting, a one-to-many form of communications (see Figure 2.5) in which information is transmitted to multiple destinations simultaneously; it utilizes the connectionless User Datagram Protocol (UDP) instead of TCP [Stevens 1994]. In the multicast communications model, all users are part of a "multicast group" in which all datagrams are sent "best effort" without any guarantee of actually being received by the distant station. An important aspect to consider is that not all participants require the same communications capabilities. Some users may wish to receive video and audio, but transmit only audio. Others may wish to receive video and/or audio only without actively participating.

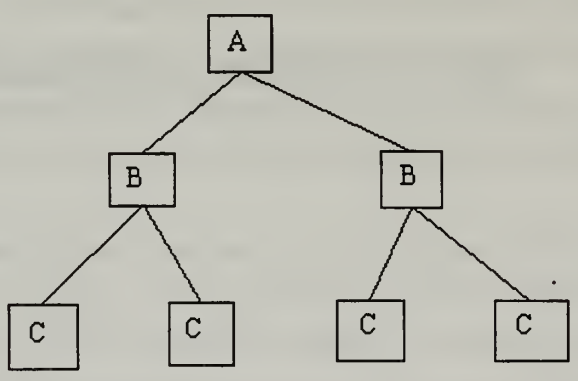


Figure 2.5: Multicast Communication Model

C. STANDARDS AND COMMERCIAL SYSTEMS

Technological improvements in networking technology, compression algorithms (codecs), microprocessors, and operating systems have made the real-time transmission of video and audio data streams between general-purpose workstations and desktop personal computers (PCs) an easily-attained application. Many legacy systems have sufficient capabilities to make real-time communications to the desktop a reality. Low-end solutions provide adequate quality for some important needs, negating the necessity for travel and generally reducing business operating costs. Several standards have been proposed or developed by the International Telecommunications Union (ITU), easing the transition to compatible applications on desktop PCs. For example, H.320 is the established standard for video teleconferencing over a WAN, providing for the coding, transmission, and decoding of video and audio data streams over ISDN lines; H.323 defines how PCs can interoperate to share video and audio data streams over both intranets and the Internet, networks that do not provide a guaranteed quality of service; G.117, G.722, G.723, G.728, and G.729 pertain to the encoding of audio data; and the T.120 series of standards provides for multimedia communications protocols, used in common collaborative tools such as whiteboards and application sharing. Older standards still exist, such as the H.200 series which consists of several specific standards for user interface, picture formatting, video and audio codecs (compression/decompression), call setup and conference management [Perloff 1993].

Many commercial platforms exist for video teleconferencing, such as those provided by PictureTel, V-Tel, and Vivo, but most of them are large conferenceroom systems with specially located microphones and ISDN connections to remote locations. Such systems are optimal for large conferences in a corporate or distance learning environment but are unnecessarily large and unsuited for lesser needs. Desktop VTC, however, provides a more suitable solution, in terms of size and money, for the user who has nothing more than a PC connected to a network.

Video teleconferencing can provide several meaningful services in the military environment to include the dissemination of intelligence data and improve coordination and performance of dispersed units under distributed control. Interactive communications shared by geographically-dispersed units will allow commanders to confer quickly, share documents and images, remotely control surveillance vehicles, and better coordinate special operations forces. In bandwidth-limited environments, commanders can eliminate

the main conferencing channel and utilize still images, maps and overlays, and an audio channel to quickly convey pertinent and time-sensitive information [Perloff 1993].

This concludes the initial presentation of VTC systems and standards. The next chapter illustrates the environment and tools used in this thesis, to include the testbed, VTC software, SNMP, and the Java programming language.

III. TESTBED NETWORK

This chapter describes the system used in the thesis. Figure 3.1 shows the overall configuration of the system used in the thesis. The Java-SNMP applet provides the graphical user interface (GUI) for interacting with the Simple Network Management Protocol (SNMP) in order to gather information from the SNMP agent residing on the managed host engaged in a VTC session. The metrics gathered by the applet are stored in a data file for analysis.

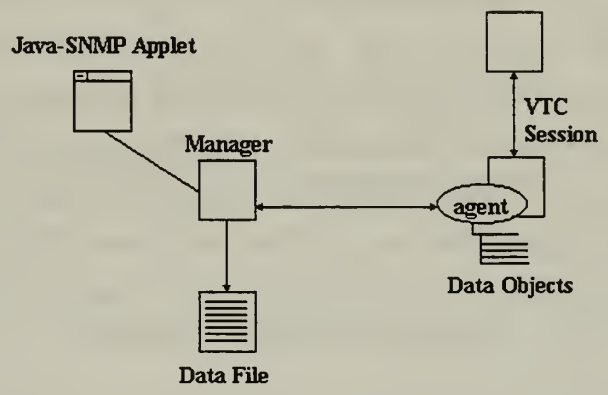


Figure 3.1: Testbed System Configuration

The following sections provide specific information on components used in this system.

A. TESTBED ENVIRONMENT

The environment in use for this thesis is a 10 Mbps Ethernet testbed network, consisting of two distinct subnetworks connected via two Cisco Systems 2514 routers, one of which is connected to the campus network at the Naval Postgraduate School (see Figure 3.2).

The Cisco routers are dual serial, dual LAN routers, each with two synchronous serial ports and two ethernet AUI ports. These routers provide suitable means for segregating the computer systems into distinct subnetworks and isolating the testbed from external traffic from the campus network. Both subnetworks operate using the IEEE 802.3 CSMA/CD LAN standard, with one subnet using 10Base-T cabling while the other uses 10Base2 cabling. Each Ethernet port has a maximum transfer rate of 10 Mbps, and

the serial ports have a maximum transfer rate of 1.544 Mbps. The routing protocol used by the routers is the Routing Information Protocol (RIP), allowing compatibility with the campus network and necessitating the use of 8-bit subnet masks. The routers communicate via a synchronous serial connection.

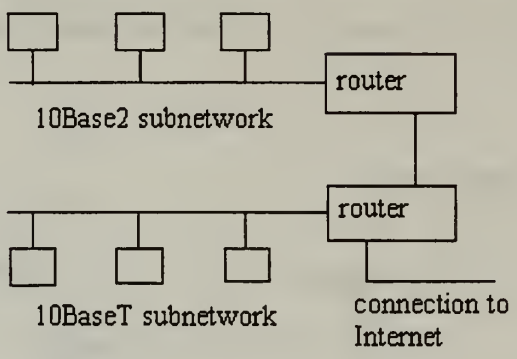


Figure 3.2: Testbed Topology

All stations on the testbed are Intel-based PC systems. Each subnet has a server running the Microsoft WindowsNT Server 3.51 operating systems, and several clients running the Microsoft Windows95 operating system.

All PCs on the testbed communicate via the TCP/IP protocol suite, which is provided along with the operating system. The layered model is shown in Figure 3.3 [Stevens 1994]. The VTC software resides at the application layer of the model, and the video and audio data are packetized by the TCP, IP, and the Ethernet layers, and then transmitted to the intended receiver.

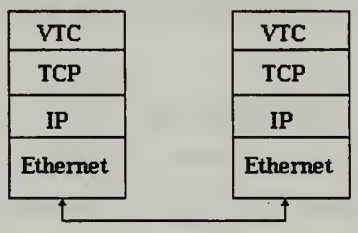


Figure 3.3: Layered Model

B. VIDEO TELECONFERENCING SOFTWARE

White Pine Software provides an excellent software-only solution for desktop VTC in Enhanced CU-SeeMe, based on the original CU-SeeMe software produced at Cornell University. Enhanced CU-SeeMe provides video teleconferencing based on the point-to-point, broadcast, and multicast communication models (see Chapter II) via TCP/IP over POTS (plain old telephone service), ISDN, LAN and the Internet, on several platforms, including Microsoft Windows and Windows95, Macintosh and Power Macintosh. Enhanced CU-SeeMe version 2.1 is used in this thesis. The software makes use of Crystal Net Corporation's Surface Fitting Method for video compression (compression ratio reported by White Pine Technical Support is 24:1) and provides audio sampling rates of 2.4Kbps, 8.5Kbps, and 16Kbps. The software receives uncompressed video and audio data from the capture source (see Figure 3.4) and compresses the data for transmission. When VTC data is received from an outside source, the software sends that data to the appropriate display device: either the monitor or the speakers.

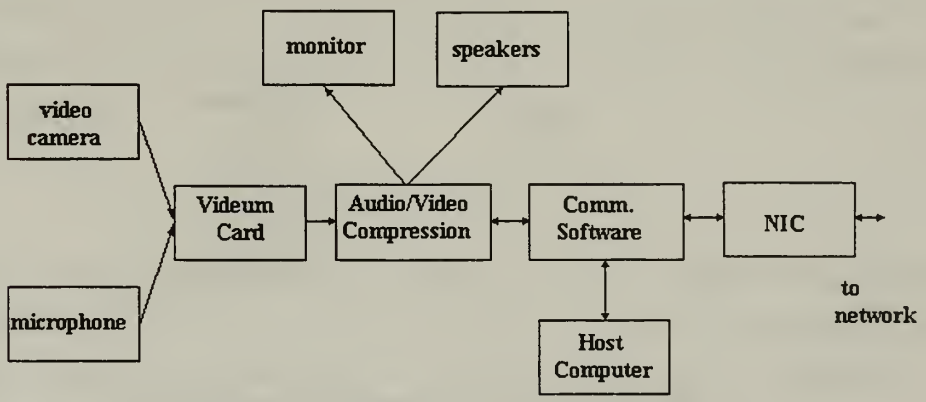


Figure 3.4: VTC System Diagram

C. VIDEO/AUDIO CAPTURE CARD

The Videum Card (see Figure 3.4), from Winnov LLP, is the video and audio capture card used in conjunction with the video teleconferencing software. The card supports a range of image sizes from 32x24 to 704x576 pixels, as well as hardware video

compression for 30 frames per second (fps) 320x240 pixels, capture-to-disk, and audio sampling rates of 8.0, 11.025, 22.05, or 44.1 KHz, in 8 and 16 bits. The video is captured via the Hitachi color camera, model KV-C25A, mounted on top of the monitor, and sent to the software for compression. The audio is relayed via a microphone headset. The drivers used in this thesis are version 1.5.2 for Microsoft Windows95, available for download from the Winnov technical support website. At the time of this writing, drivers are also available for Microsoft Windows3.1 and NT4.0.

D. SIMPLE NETWORK MANAGEMENT PROTOCOL

The Simple Network Management Protocol (SNMP) is used to gather relevant data from a host computer on the testbed. SNMP was adopted as a standard in 1989 to provide an immediate, short-term means of using the existing TCP/IP protocol infrastructure to monitor and manage devices attached to a network. SNMP is a set of standards for network management consisting of a protocol, a database structure specification (Structure of Managed Information, or SMI), and a set of data objects (Management Information Base, or MIB) (see Figure 3.5).

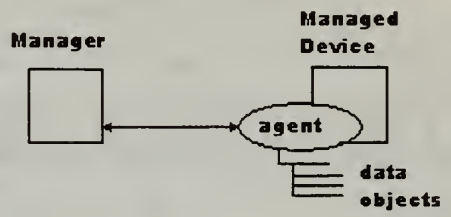


Figure 3.5: SNMP Communication Model

The protocol for SNMP version 1 is specified in Request for Comments (RFC) document 1157. RFC 1155 defines the SMI, and RFC 1213 describes the managed objects contained in the management information base for network management of TCP/IP-based Internets, or MIB-II. Other RFCs define extensions to the SMI or MIB [Stallings 1996]. SNMP resides at the application layer of the IP layer model and makes use of the connectionless user datagram protocol (UDP) transport mechanism (see Figure 3.6) to issue requests for the status of specific data objects maintained by the agent on the

managed device (see Figure 3.5). These requests are encoded per the Basic Encoding Rules (BER) associated with ISO Abstract Syntax Notation One (ASN.1), issued to the agent on UDP port 161, and the responses are received on UDP port 161 and decoded into human-readable form [Ben-Artzi 1990], [Stallings 1996]. Several commercial products are available to provide the network manager with a simple means of issuing requests for reasonably large networks and for extracting meaningful information germane to the health and status of devices on the network.

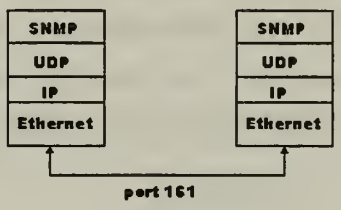


Figure 3.6: Layered Model with SNMP

SNMP version 1 and MIB-II, based on RFCs 1157 and 1213, respectively, are widely supported and easily accessed on all systems on the testbed. Information on octet rates into and out of the network interface of the host computer was easily gathered using a simple Java applet.

E. JAVA

Java is an object-oriented, platform-independant, interpreted programming language developed by Sun Microsystems. Java allows for the development of either complete applications or application stubs (applets) that require a web browser or other suitable environment in which to operate ([Cornell 1996], [Flanagan 1996]), and is used in conjunction with the Java SNMP package from Advent Network Management, Inc, to provide the framework for developing an applet to facilitate gathering SNMP information from the host computer polled in this thesis. The applet polls the host computer for data objects specific to MIB-II, at a user-defined interval (as detailed in Chapter V). Java allows for cross-platform development, in that any platform supporting the Java Virtual Machine will be able to run the applet written for this thesis (Appendices A and B). The development environment used in this thesis is Symantec Cafe 1.51, based on the Sun

Microsystems Java Development Kit version 1.0.2, running on Microsoft WindowsNT Server 3.51.

Two Java applets were developed for this thesis: SNMP and video source simulation. The former is described here and the latter in Chapter IV.

The Java-SNMP applet (see Appendix A for source code) has a simple flow (see Figure 3.7). The applet was developed from a combination of standard Java classes and classes specifically designed for SNMP. When the applet is first initialized by typing the command "appletviewer MibCrusher.html" from the user prompt (DOS or Unix), the graphical user interface (GUI) is available with several default settings; the IP address of the host to be polled, the filename of the data file, and the number of seconds delay desired between polls. These values can be easily changed by the user (see Figure 3.8).

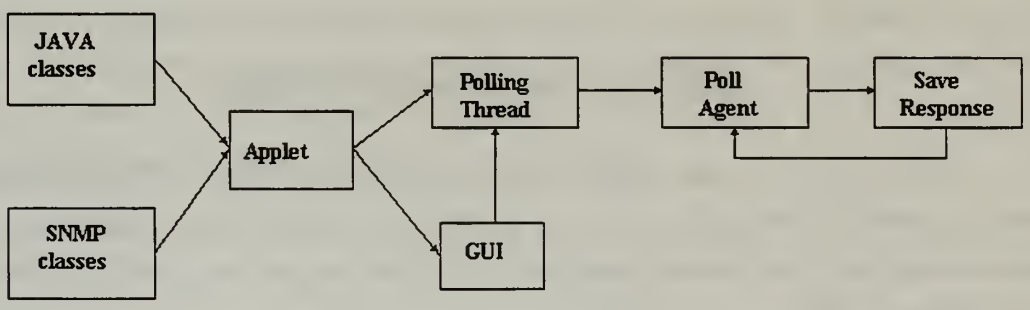


Figure 3.7: Java-SNMP Applet Flow Diagram

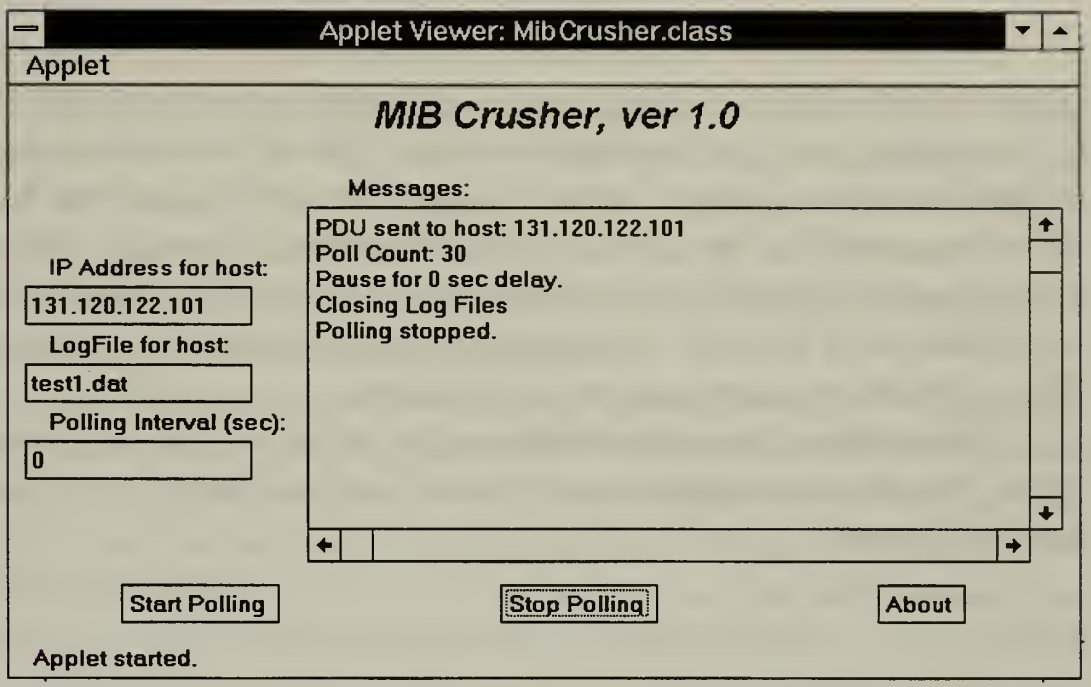


Figure 3.8: Graphical User Interface for Java-SNMP Applet

The testing period begins following successful connection and communication with a remote host running the video teleconferencing system. Once communication is established, the "Start Polling" button on the GUI of the application is clicked to initiate the polling thread of the applet and open the data file. The polling thread issues get-request process data units (PDUs) to the agent on the managed device (see Figure 3.5) and writes the response, along with a time stamp, to the data file. This thread continues until it is interrupted by the user by clicking on the "Stop Polling" button, at which point the data file is closed. From this point, the user can immediately initiate another polling session.

The first test was conducted on the testbed using the point-to-point communication model, in order to gather data during a video-only session. The session consisted of "talking heads" in which the users on either system kept extraneous movements to a minimum while moving only their heads and lips, thus simulating a video teleconferencing session without the audio component. Throughout this session, the agent on the managed host was polled in excess of 10,000 times, with the delay between polls set to 0 seconds. The average time between polls for this test was 145.6 milliseconds. The returned data was stored in the "v1114.dat" logfile.

The second test was conducted across the campus network, with a remote system in another building again using the point-to-point communication model. This allowed for a more realistic setting, eliminating the annoyance of the audio echo that occurs while teleconferencing with a "remote" system in the same room. This test used both the video and audio capabilities of the video teleconferencing system. Again, the agent on the managed host was subjected to more than 10,000 get-request PDUs during the session, with the delay set to 0 seconds. The average time between polls was 111.2 milliseconds. The returned data was stored in the "va1114.dat" logfile.

This concludes the presentation of the testbed environment and tools used in this thesis. The next chapter discusses self-similarity and the statistical models used to simulate VBR traffic.

IV. TRAFFIC MODELING

This chapter discusses self-similarity and the model used to simulate a single VBR video source. A VBR video source, such as a video teleconferencing application, produces network traffic that is self-similar in nature [Likhanov 1995]. A model is developed in order to simulate a VBR video source, and the self-similar nature of the simulated traffic is verified. This model is then expanded to include an audio source in order to simulate the aggregate audio and video traffic.

Whenever planning for the installation of a computer network, or the upgrade of an existing network, especially one of appreciable size, it is instructive to have some idea as to the traffic models to be encountered. Traffic models are especially useful in developing congestion and flow control techniques, as is being done in the case of Broadband ISDN (B-ISDN or ATM). Having some idea as to the traffic characteristics that will likely be encountered on a network is particularly useful when a simulation tool is used to first model the network and then simulate anticipated changes to the topology. The effects of possible disruptions to the network can then be observed. As more and more LANs are being interconnected via broadband networks, accurate traffic models are required to produce meaningful admission, access, and flow control mechanisms.

A. SELF-SIMILAR NATURE OF TRAFFIC

Though the actual information traveling across a computer network is nothing more than electrical impulses representing 1s and 0s, data traffic is distinctly different from multimedia traffic. Data traffic is predominantly bursty and requires reliable service from the network, though it will tolerate delays. However, multimedia traffic is interactive and stream-oriented, in that a continuous flow of information is expected to be delivered within a bounded delay of 100-200 msec, and some data loss is tolerated. Uncompressed audio and video data travel across the network at a constant bit rate, whereas compressed data has a variable bit rate nature. Most existing network infrastructures, however, were originally developed with data applications in mind and do not provide the necessary bounded delay and loss required of interactive, multimedia communications [Dilgac 1994].

The study and development of traffic models are ongoing areas of research in the context of B-ISDN, especially for the types of traffic that are expected to utilize the B-ISDN transport mechanism, but a thorough examination of the Ethernet traffic has

already been done [Leland 1994]. Standard models for network traffic, such as pure Poisson and Markov-modulated Poisson processes, provide an inaccurate view of the nature of the Ethernet traffic [Subramanian 1995]. Statistical analysis of high-quality, high-resolution Ethernet LAN traffic data shows the self-similar or fractal nature of traffic, behavior that is markedly different from conventional telephone traffic and from generally accepted models. The term "self-similar" refers to the characteristic in which the traffic appears to be similar, invariant of the time scale at which the traffic is observed. That is, traffic plots retain similar periods of burstiness regardless of the time resolution in which the traffic is viewed. The graphical representation of self-similar traffic lacks a natural burst length regardless of the number of aggregated sources whereas the traffic plot derived from traditional models smooths out as aggregation of sources increases. Traffic plots for currently considered stochastic models become indistinguishable from white noise when aggregated over even a few hundred milliseconds [Leland 1994].

In order to define self-similarity [Leland 1994], let sequence X be a covariance stationary stochastic process with mean μ, variance σ^2, and autocorrelation function $r(k)$, $k \geq 0$. From this sequence, a new covariance stationary time series, $X^{(m)}$, is obtained by averaging the original series over non-overlapping blocks of size m. For each m, $X^{(m)}$ is given by

$$X^{(m)}_k = (1/m)(X_{km-m+1} + X_{km-m+2} + \ldots + X_{km}), k \geq 1$$

The original sequence X is called (exactly) second-order self-similar if for all m = 1, 2, ...,

$$\mathrm{var}(X^{(m)}) = \sigma^2 m^{-\beta}, 0 < \beta < 1$$

and

$$r^{(m)}(k) = r(k), k \geq 0$$

The original sequence is (asymptotically) second-order self-similar if, for all given m, the corresponding aggregated processes $X^{(m)}$ are the same as or become indistinguishable from X, with respect to their autocorrelation functions.

An important characteristic that differentiates second-order self-similar process from typical traffic models considered in current literature is the presence of a nondegenerate correlation structure for the aggregated processes $X^{(m)}$ as $m \Rightarrow \infty$. The correlation structure of the aggregated processes $X^{(m)}$ for typical traffic models tends toward second-order white noise:

$$r^{(m)}(k) \Rightarrow 0, \text{ as } m \Rightarrow \infty, \text{ for } k \geq 1$$

It is in the context of B-ISDN that these characteristics are of particular importance as the nature of congestion produced by self-similar traffic is different and significantly more complicated than the standard formal models accepted to date. A B-ISDN network is expected to support many classes of traffic, including variable bit rate (VBR) traffic produced by video teleconferencing systems. In order to regulate congestion on such a network, research needs to be done in the area of access control and particularly accurate models will be required, without which a proposed access control scheme will be somewhat less than reliable [Likhanov 1995].

B. MINISOURCE

Video traffic is expected to dominate the bandwidth of future broadband networks. The most basic representation of VBR video traffic is an equivalent process based on a sum of identical two-state on-off sources. The two-state on-off model is shown in Figure 4.1 [Schwartz 1996].

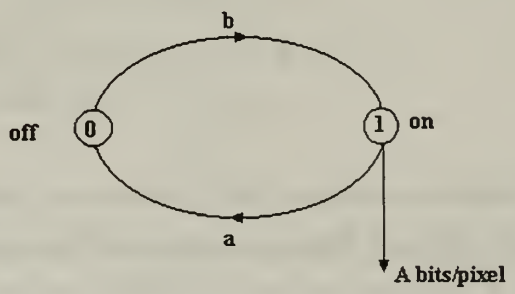


Figure 4.1: Two-State On-Off Model

The model, or minisource, moves back and forth exponentially between the "off" and "on" states. While in the "on" state, A bits/pixel are offered by the process. A total

of M minisources are concatenated to form a continuous-time, discrete state Markov chain, representing a video source (see Figure 4.2). The resultant model is a two-state doubly stochastic Poisson process with exponential sojourn times, given by parameters a and b. This model is fully capable of capturing the self-similar characteristics of video traffic [Subramanian 1995]. Other models that have been mathematically proven to exhibit self-similar characteristics are a superposition of infinity minisources with Pareto service demands [Likhanov 1995] and a fractional Gaussian noise source [Cinotti 1995].

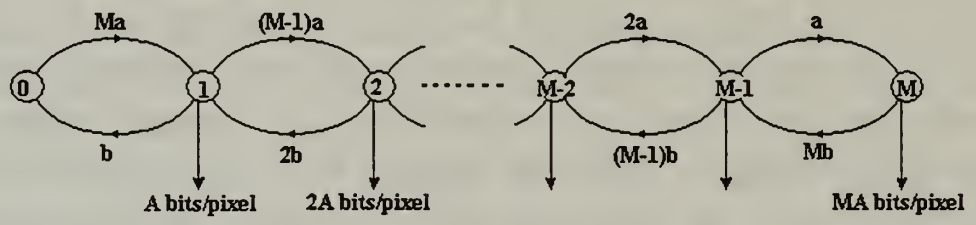


Figure 4.2: Markov Chain Representation of VBR Source

The Markov chain in Figure 4.2 is an equivalent process defined to be the sum of M two-state minisources with exponential sojourn times a and b, in which A bits/pixel are transmitted while in the "on" state. The composite process is represented by an (M+1)-state Markov chain. The transmission rate of the composite process is quantized to the levels 0, A, 2A, ..., MA bits/pixel. The three minisource model parameters a, b, and A are given by [Schwartz 1996]

$$b = 3.9/(1 + 5.05\ N/M)$$
$$a = 3.9 - b = 19.7\ N/M(1 + 5.05\ N/M)$$
$$A = 0.4/b = 0.1 + 052\ N/M$$

The composite process matches the M-minisource model to N multiplexed video sources. Given that M=20 and N=1, we have a=0.78, b=3.12, and A=0.13 bits/pixel. Each quantized level is characterized by 7339.296M bits (or 924.9M bytes) transmitted during the sample time slot. The time slot is determined by approximating the time between samples while gathering data from the testbed. For the purpose of simulation, the time slot is set to 145 msec, which is the average time between polls for the video-only traffic.

C. SIMULATION

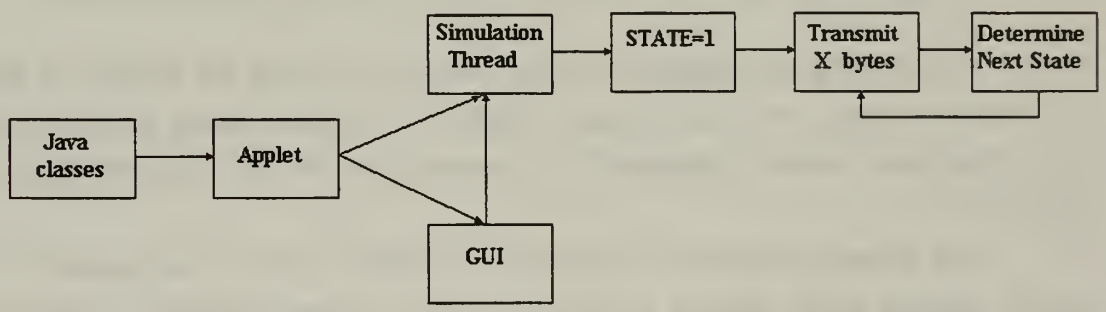
The model has predetermined values for the transitions between states (Ma, (M-1)a, ..., a, and b, 2b, ..., Mb) based upon the values of a and b. The model begins at STATE=0, and proceeds to STATE=1, then to STATE=2, etc. At each state, two exponentially distributed random variables are computed [Leon-Garcia 1994]:

$$X_1 = (-1/a) \ln(U_1)$$

$$X_2 = (-1/b) \ln(U_2)$$

where X_1 and X_2 are the exponentially distributed random variables with parameters a and b, and U_1 and U_2 are uniformly distributed in [0,1]. The values a and b are transition rates from state[i] to state[i+1] and state[i-1], respectively; and the decision of which state to transition to is made based upon the smaller of the two random variables, X_1 and X_2. At each state, the time stamp and the number of bits transmitted are recorded.

A Java applet to model the VBR video source in Figure 4.2 has been developed. The algorithm consists of a concatenation of twenty minisources into a Markov chain representation of the VBR source. The program flow of the applet can be seen in Figure 4.3. The source code for the applet is included in Appendix B.



4.3: Video Source Simulation Applet Flow Diagram

The simulation applet is invoked using the command "appletviewer SimTest.html" at the user prompt (DOS or Unix). After the GUI (see Figure 4.4) is displayed, the user starts the simulation thread by clicking on the "start" button, and halts the thread by clicking on the "stop" button. The "close" button disposes of the applet.

The logfile into which the simulation data is saved is determined by the user, and is set to "sim1.dat" by default.

When the simulation thread is started, the logfile is opened, and the initial state of the Markov chain is 0. The simulation then proceeds to state 1 and the time stamp and A bytes are recorded in the logfile. The simulation algorithm must then determine into which state to proceed by computing two exponential random variables and moving in the direction of the lower value. At each state, the number of bytes and the time stamp are recorded in the logfile.

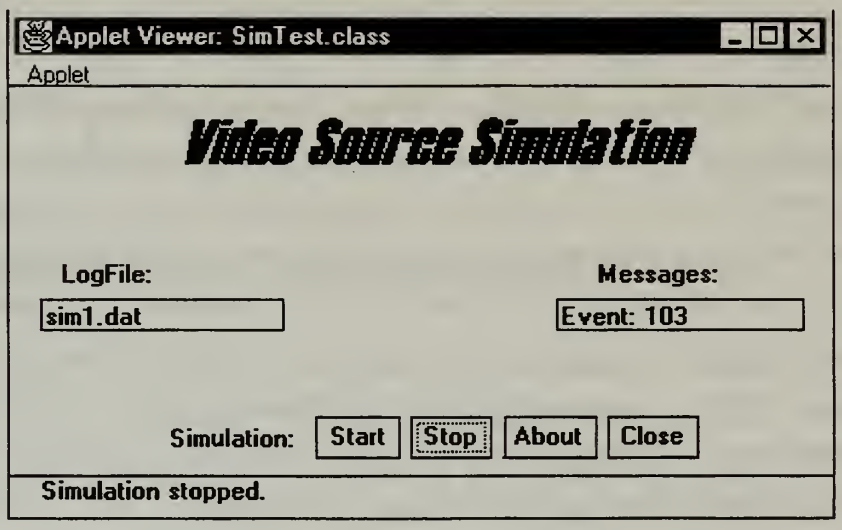


Figure 4.4: Graphical User Interface for VBR Simulation Applet

The video source simulation applet was extended with the addition of an audio source simulation in order to produce an aggregate simulated traffic source (see Figure 4.5). The audio source is simulated in accordance with the minisource model shown in Figure 4.1.

The source code for the audio/video simulation applet is in Appendix C. The applet operates in an identical manner to the video source simulation applet, with the addition of an audio model to simulate the audio component. A program thread for the audio source runs concurrently with the thread for the video source. When the video source is sampled, the audio source is sampled as well. The parameters for the audio source model are chosen such that the average silent interval (1/a) is 0.6 sec, and the average talk spurt (1/b) is 0.4 sec [Schwartz 1996]. Each sample returns the number of bytes generated during the talk spurt of the audio source.

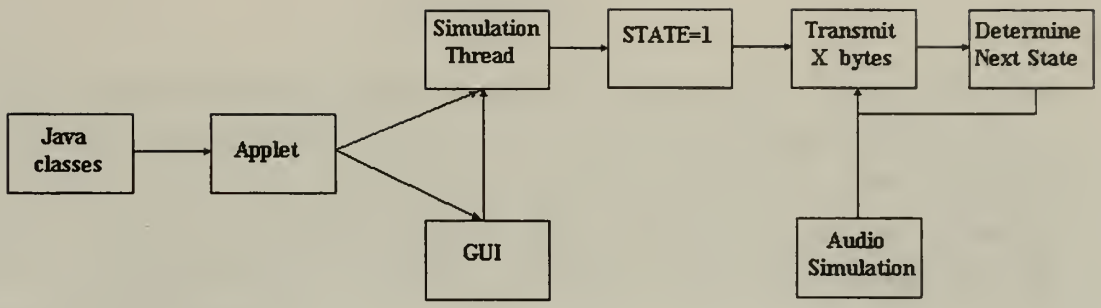


Figure 4.5: Audio/Video Simulation Applet Flow Diagram

The data recorded from the audio/video source simulation applet is stored in the logfile "sim2.dat", in the same manner as the video source simulation applet. For the purpose of simulation, the time slot is set to 110 msec, which is the average time between polls for the measured audio/video traffic.

This concludes the discussion of self-similarity and statistical models used to simulate VBR traffic. The next chapter illustrates and discusses the results of graphical analysis of measured and simulated traffic, for both video-only and aggregate video/audio traffic.

V. RESULTS

This chapter describes the results of graphical analysis of the data obtained from the testbed (measured data) and from the video source simulation applet (simulated data). The measured video/audio data is an aggregate of video and audio traffic and is displayed along with data from a video/audio simulation applet in order to illustrate the self-similar nature of the aggregate traffic. The results presented in this chapter illustrate the self-similar nature of VTC traffic.

A. DATA COLLECTION

The purpose of data gathering in this thesis is to obtain specific information on the traffic generated by a video teleconferencing application in a packet-switched environment. To this end, it was decided that SNMP would be used as it is readily available on systems utilizing the TCP/IP protocol suite, and it required no additional hardware and only a modest amount of additional software. Specifically, the management information base (MIB-II) variable "interfaces.ifTable.ifEntry.ifInOctets" (fully-qualified object identifier 1.3.6.1.2.2.10 [Stallings 1996]) was continually polled throughout the testing period, using a customized Java applet (Chapter III). Each time the applet issues a get-request protocol data unit (PDU) to the agent on the managed station, it obtains the current system time (in milliseconds). This timestamp is then written to a data file in columnar format along with the information provided in the get-response PDU returned by the agent on the managed station. This format made the data file especially easy to access using MatLab (see Appendix C), which was used for display and analysis of the data.

Host computers used in this thesis utilized hardware and software as outlined in Chapter III. All systems involved in data gathering utilized the same Enhanced CU-SeeMe conference settings:

Maximum Transmission Rate	80 kbps
Maximum Receive Rate	300 kbps
Image Size	320×160 pixels

Table 1: Common Enhanced CU-SeeMe Conference Settings

B. RESULTS

Data gathered during the video teleconferencing sessions and the simulation was analyzed with the assistance of MATLAB [MathWorks 1992]. The MATLAB ".m" files used for this purpose are listed in Appendix C. The basis for the analysis of the traffic data is given by [Leland 1994], in which statistical and graphical tools are used to not only establish the self-similar nature of the traffic, but also determine the degree of the self-similarity of the traffic via an estimate of H, the Hurst parameter. This thesis uses similar graphical methods to determine the self-similar nature of the traffic, which is represented as a data sequence within logfiles generated by Java applets (Chapters III and IV). These methods are the rescaled adjusted range statistic (R/S statistic), the variance-time curve, and the index of dispersion for counts (IDC). For the purposes of this thesis, the analysis of the data will be conducted only on the traffic that arrived at the host computer interface, which corresponds to the ifInOctets data object, from both tests. A detailed description of the Hurst parameter and how to determine it can be found in [Leland 1994] and is beyond the scope of this thesis.

The first video teleconferencing session was conducted to gather data on video-only traffic (see Chapter III) using the point-to-point communication model. Figure 5.1 is a graphical representation of the video-only traffic received by the computer interface, in kilobytes per second (kBps) versus time in milliseconds. The data was gathered by the Java-SNMP applet (Chapter III). This plot is different from traffic generated by the traditionally accepted models for telephone and packet traffic. These conventional stochastic models, be they compound Poisson or Markov processes, generate traffic plots that are indistinguishable from white noise when viewed on scales from tens to hundreds of seconds [Leland 1994].

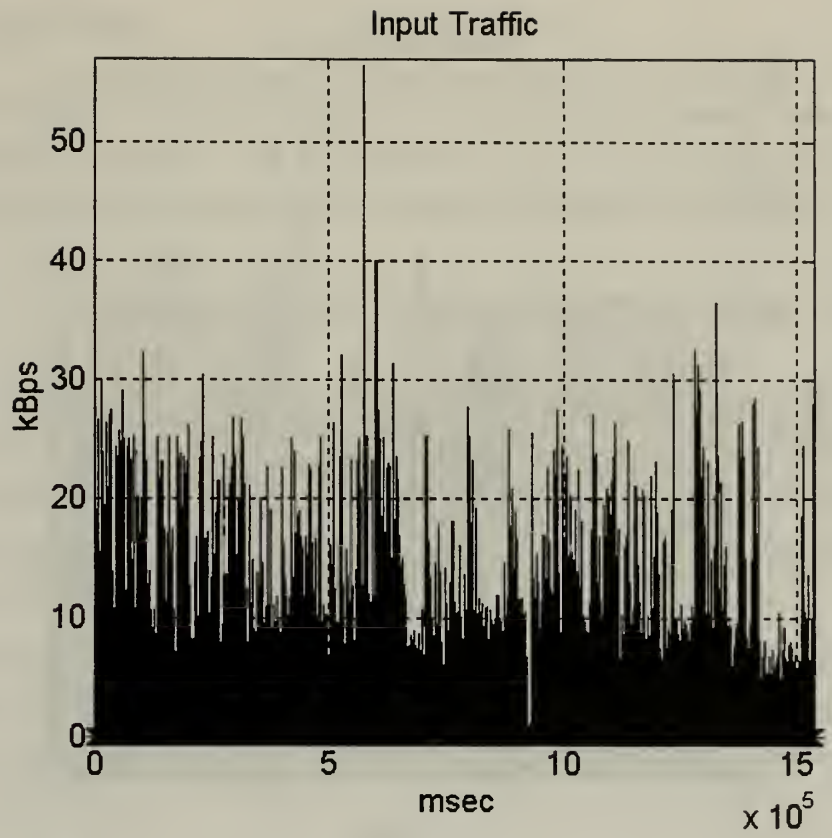


Figure 5.1: Input Traffic Into Computer Interface, Video Only

Figure 5.2 is a graphical representation of the traffic generated by the video source simulation Java applet. The simulated video traffic data is generated for the same number of data points as that in Figure 5.1. Our goal here is to apply the same metrics to both measured (Figure 5.1) and simulated (Figure 5.2) data to illustrate that, whether measured or simulated, the video data exhibits self-similar characteristics.

The traffic information from both data sets is composed of a data sequence of 10000 data points. The same number of data points are used in order to make a relevant comparison between the measured and simulated data.

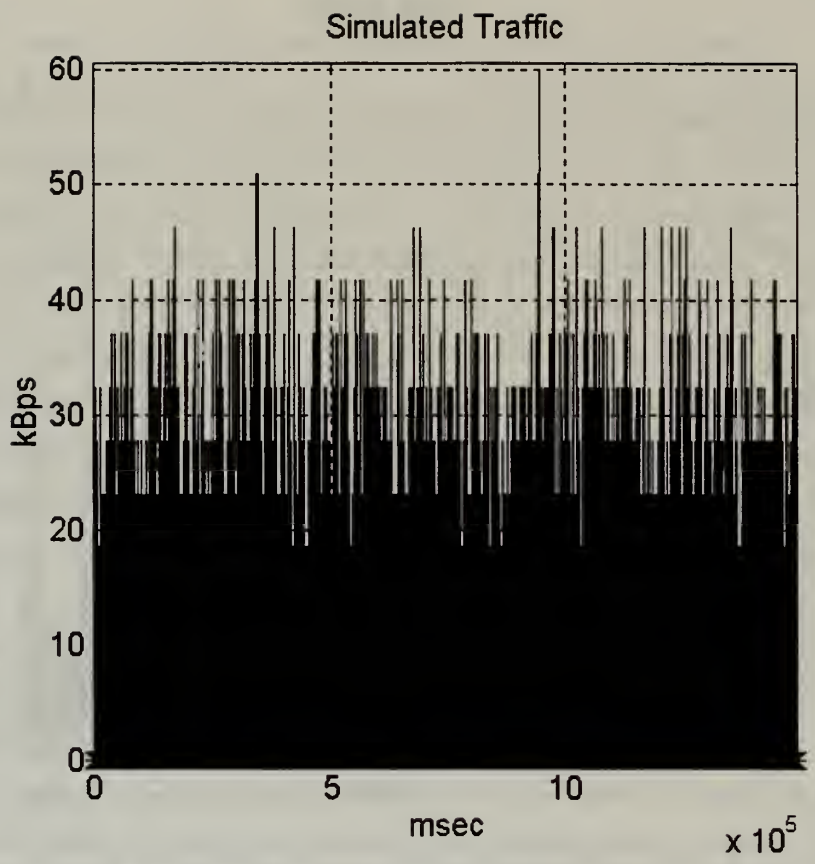


Figure 5.2: Simulated Traffic

Each sample sequence of 10000 data points, X, is considered to be a covariance stationary stochastic process [Leland 1994] with mean μ, variance σ^2, and autocorrelation function $r(k)$, $k \geq 0$. From this sequence, a new covariance stationary time series, $X^{(m)}$, is obtained by averaging the original series over non-overlapping blocks of size m. For the purposes of this thesis, m = [10 20 50 100 200 500 1000 2000 5000 10000], where m = 10000 represents the original sequence. The original sequence can then be called self-similar if, for all given m, the corresponding aggregated processes $X^{(m)}$ exhibit the same characteristics as or become indistinguishable from X (see Chapter IV).

Self-similarity can be determined based on several metrics. In the following, three such metrics are used to numerically measure the self-similar nature of both the collected and simulated data described above.

1. R/S Statistic

The rescaled adjusted range statistic, or R/S statistic, is used to infer the degree of self-similarity of a process. The R/S statistic for a given data sequence (X_k: k = 1, 2,..., n), with mean μ_n and variance $\sigma^2(n)$, is computed as follows [Leland 1994]:

$$R(n)/S(n) = 1/\sigma\ [\max(0, W_1, W_2, ..., W_n) - \min(0, W_1, W_2, ..., W_n)]$$

where $W_k = (X_1 + X_2 + ... + X_n) - k\mu_n$ ($k \geq 1$) and R(n)/S(n) refers to the R/S statistic as a function of n. A typical rescaled adjusted range plot (plot of the R/S statistic) starts with a transient zone but eventually settles down and fluctuates about a particular asymptotic slope. When plotted on a logarithmic scale as a function of *m*, the asymptotic slope of the R/S values is between 1/2 and 1 for self-similar processes [Leland 1994].

Figure 5.3 is a plot of the R/S statistic for the video-only traffic. The traffic displays an asymptotic slope that is clearly between 1/2 and 1 (lower and upper dashed lines, respectively), illustrating that the VBR traffic measured during the video-only test is self-similar in nature.

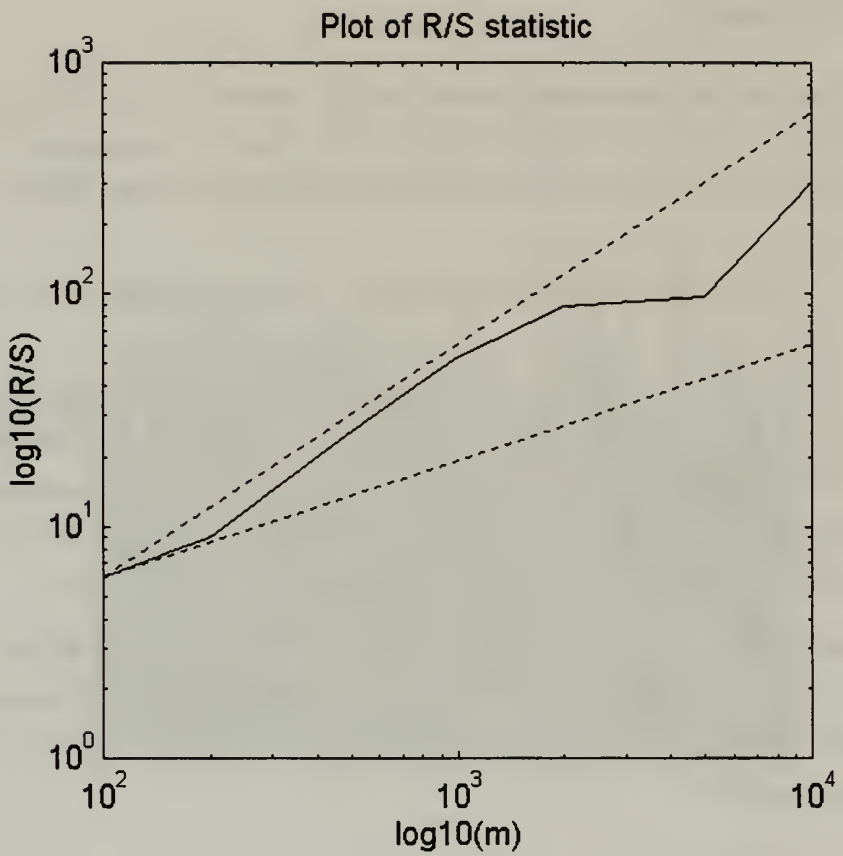


Figure 5.3: Plot of R/S Statistic, Video-Only Traffic (Measured)

Figure 5.4 is a plot of the R/S statistic for the simulated traffic, shown in Figure 5.2. This traffic also displays characteristics of self-similarity based on the R/S statistic as the asymptotic slope of the R/S statistic data falls between 1/2 and 1 (lower and upper dashed lines, respectively).

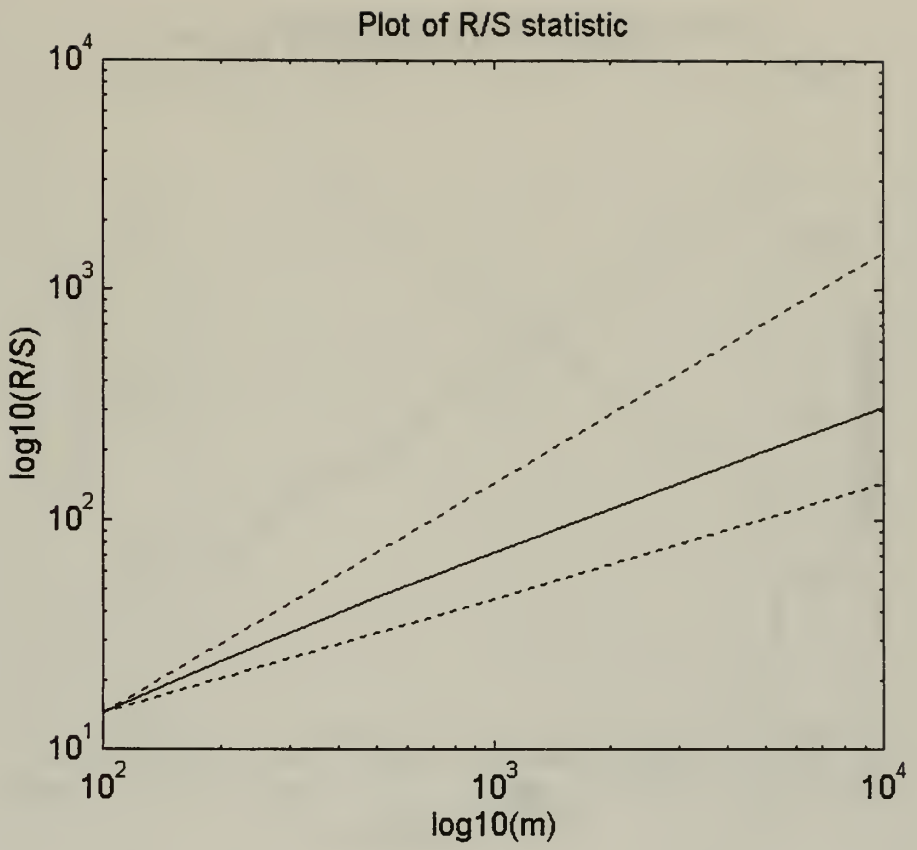


Figure 5.4: Plot of R/S Statistic, Video-Only Traffic (Simulated)

2. Variance-Time Curve

When the variances of the aggregated sequences $X^{(m)}$ are plotted on a logarithmic scale as a function of m, the asymptotic slope of the data points is distinctly different from -1. Values for the asymptotic slope between -1 and 0 suggest self-similarity [Leland 1994].

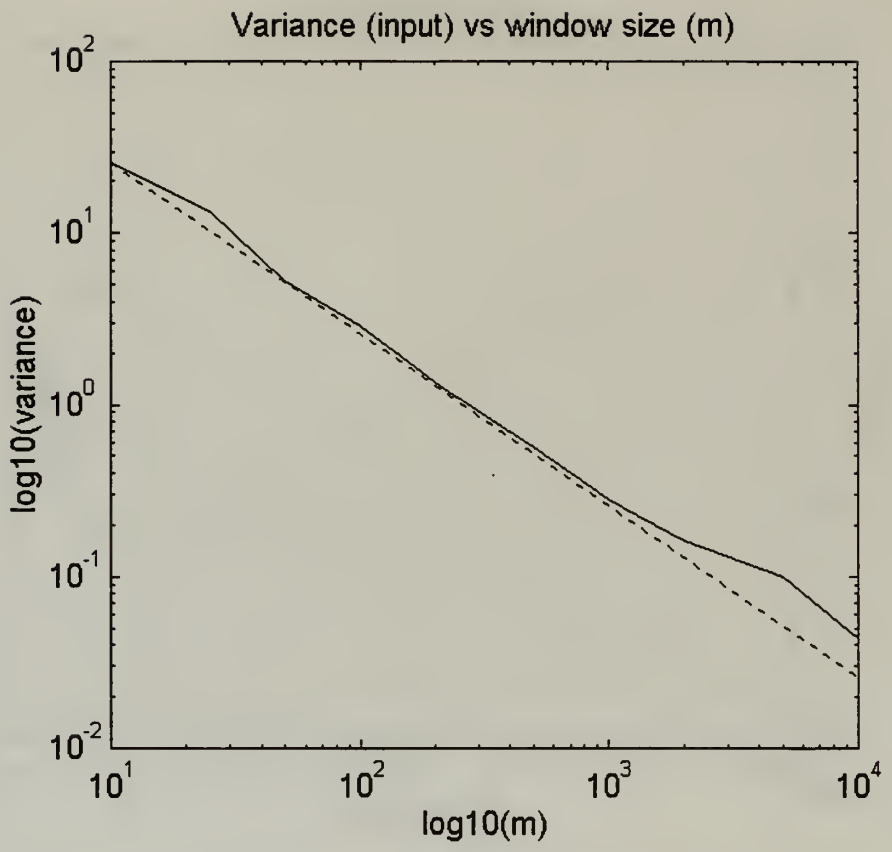


Figure 5.5: Variance-Time Curve, Video-Only Traffic (Measured)

The variance-time curve for the video-only traffic is displayed in Figure 5.5. The asymptotic slope of the data is greater than -1, indicating the self-similar nature of the data. However, the slope is not significantly greater than -1 (dashed line has slope of -1), as illustrated in [Leland 1994], suggesting that the data may not be quite as self-similar as first supposed. However, this metric is not solely employed to determine the degree of self-similarity of the traffic, but rather simply to illustrate that the data has self-similar characteristics. The variance-time curve in Figure 5.6 illustrates that simulated traffic is self-similar in nature, in the same manner as was illustrated for Figure 5.5 (dashed line has slope of -1).

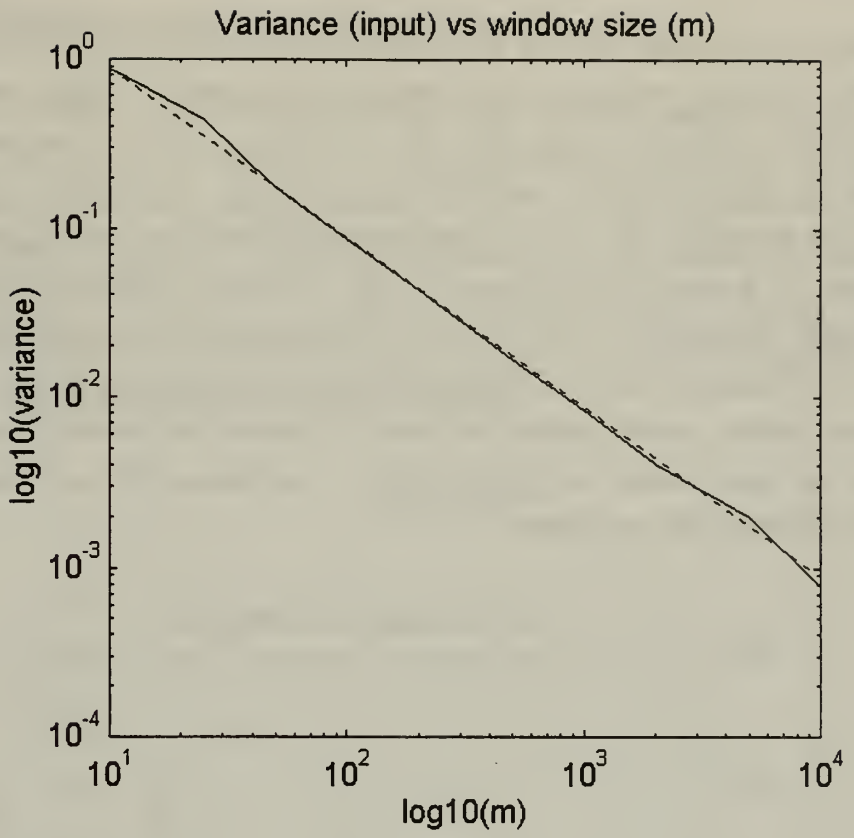


Figure 5.6: Variance-Time Curve, Video-Only Traffic (Simulated)

3. Index of Dispersion for Counts (IDC)

The *index of dispersion for counts* (IDC) is a popular measure used to capture the variability of traffic over different time scales [Cinotti 1995], [Fowler 1991], [Leland 1994], and [Subramanian 1995]. For a given time interval of length L of sequence X, the IDC is given by:

$$\text{IDC}(L) = \text{var}(X(L)) \,/\, \text{E}[X(L)]$$

Stated simply, the IDC for a sequence of length L is the ratio of the variance of the number of arrivals during the interval to the expected value of the number of arrivals during that same interval.

The IDC is plotted on a logarithmic scale as a function of L, as seen in Figure 5.7. For self-similar processes, the IDC increases monitonically throughout the time span.

Conventional traffic models, such as pure Poisson processes, have an IDC equal to one. Other arrival models, including batch Poisson, deterministic batch, and Markov-modulated Poisson processes, have IDCs that converge on fixed values over the time scales observed [Fowler 1991]. The IDC curve provides an immediate, engineering-based approach to testing a set of traffic measurements for self-similarity [Leland 1994].

Figure 5.7 presents the IDC curves for the video-only traffic (solid line) and the simulated traffic (dashed line). The curves are monitonically increasing, indicating that the traffic is self-similar in nature. The observed gap between the curves is a result of the characteristics of the specific traffic data used in deriving the curves (Figures 5.1 and 5.2). While the monitonically increasing behavior is maintained, the curve could shift up or down for different video sequences.

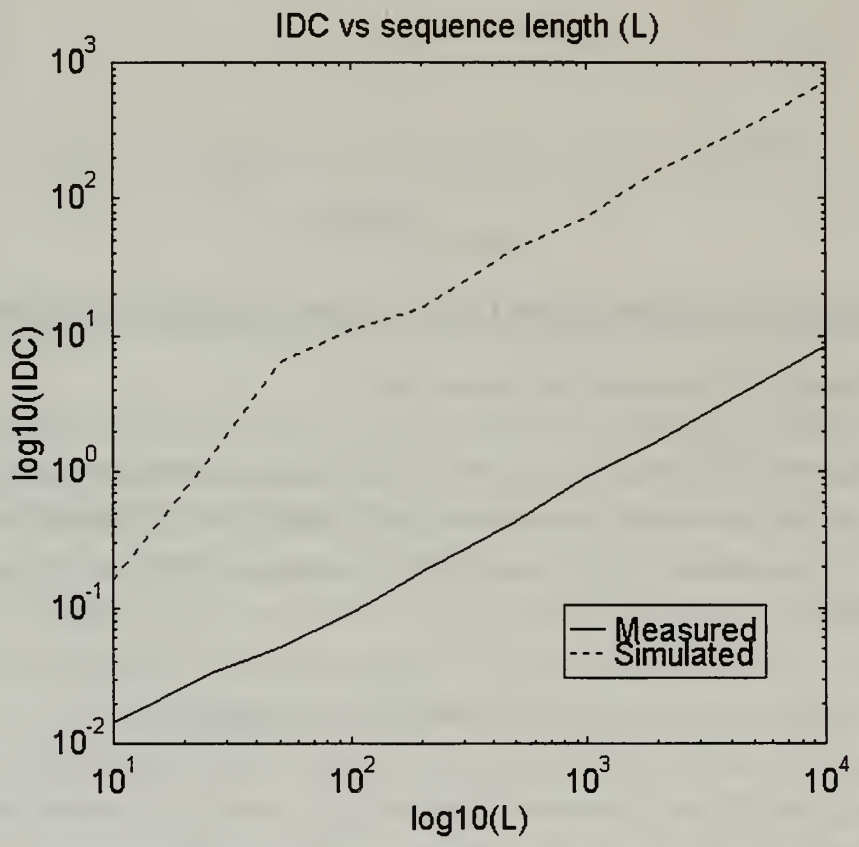


Figure 5.7: IDC Curves for Video-Only Measured and Simulated Traffic

C. RESULTS FOR VIDEO/AUDIO TRAFFIC

Figure 5.8 shows the graphical representation of traffic gathered during a video teleconferencing session in which both the video and audio components of the system are used. The input traffic at the computer interface (NIC) is an aggregation of the video and audio traffic generated during a video teleconferencing session in which both users send and receive video and audio data. In this session, conditions similar to those in the video-only case were applied, in that both users kept extraneous head and hand movements to a minimum, giving the session a "talking heads" quality.

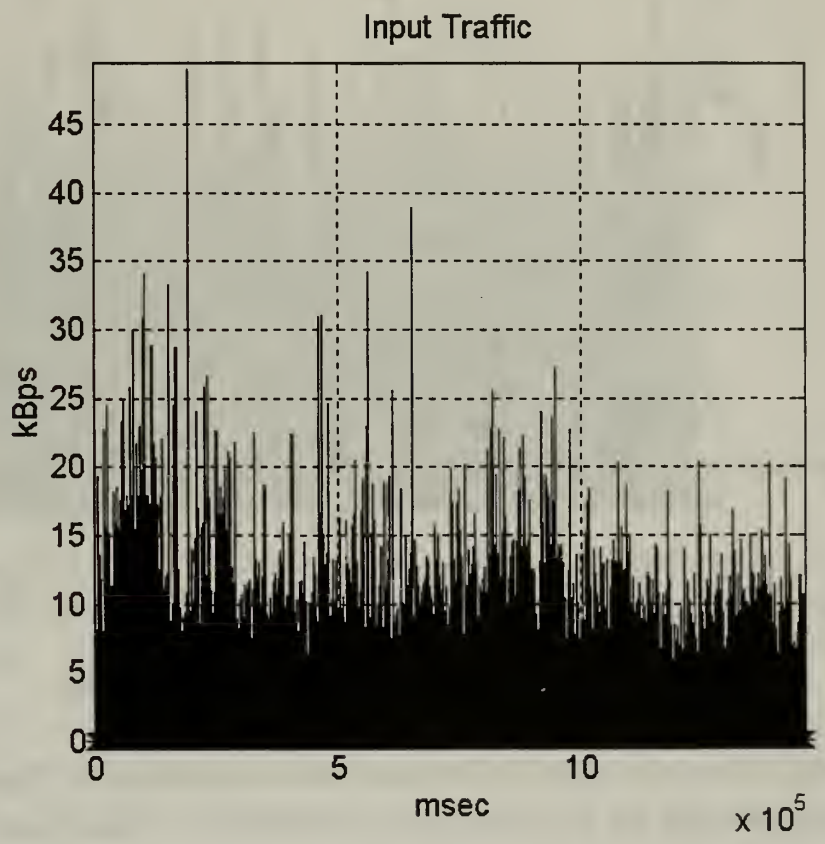


Figure 5.8: Input Traffic at Computer Interface, Video/Audio

Figure 5.9 is a graphical representation of the traffic generated by the video/audio source simulation applet. The simulated traffic data is generated for the same number of data points as those in Figure 5.8. Our goal here is to apply the same metrics to both

measured (Figure 5.8) and simulated (Figure 5.9) data to illustrate that, whether measured of simulated, the video/audio data exhibits self-similar characteristics.

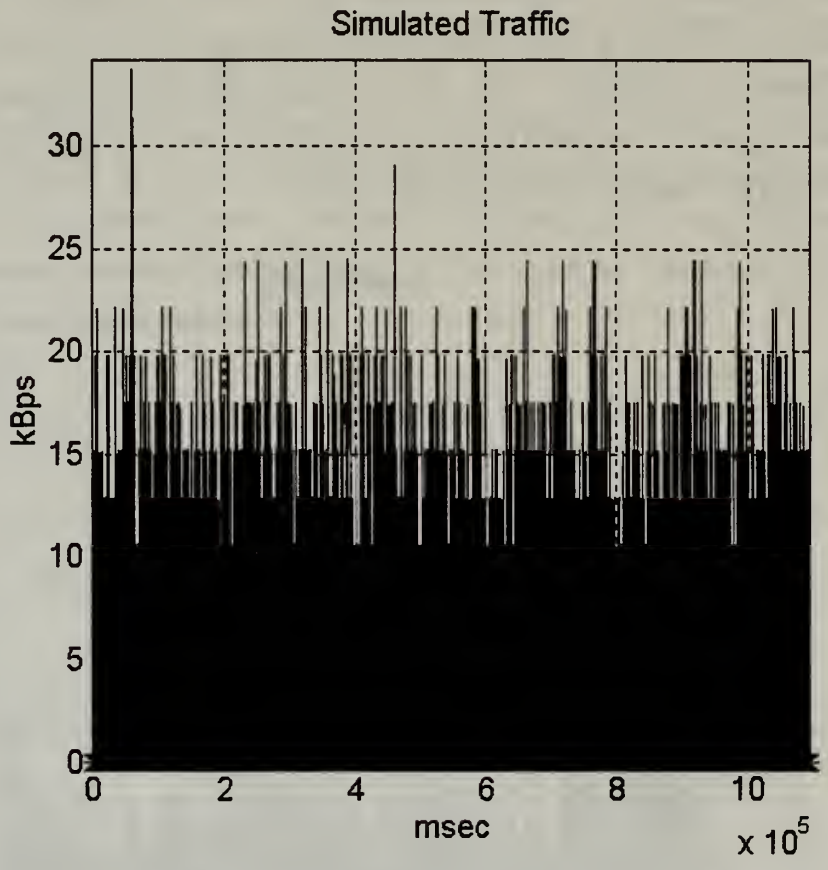


Figure 5.9: Simulated Audio/Video Traffic

Figure 5.10 is a plot of the R/S statistic for the traffic shown in Figure 5.8. The plot of the R/S statistic for the video/audio shows similar characteristics to the R/S statistics plotted in Figures 5.3 and 5.4, indicating in the same manner that the video/audio traffic is also self-similar in nature. The asymptotic slope of the R/S statistic data in Figure 5.10 falls between 1/2 and 1 (lower and upper solid lines, respectively), illustrating that the aggregate video/audio data is also self-similar.

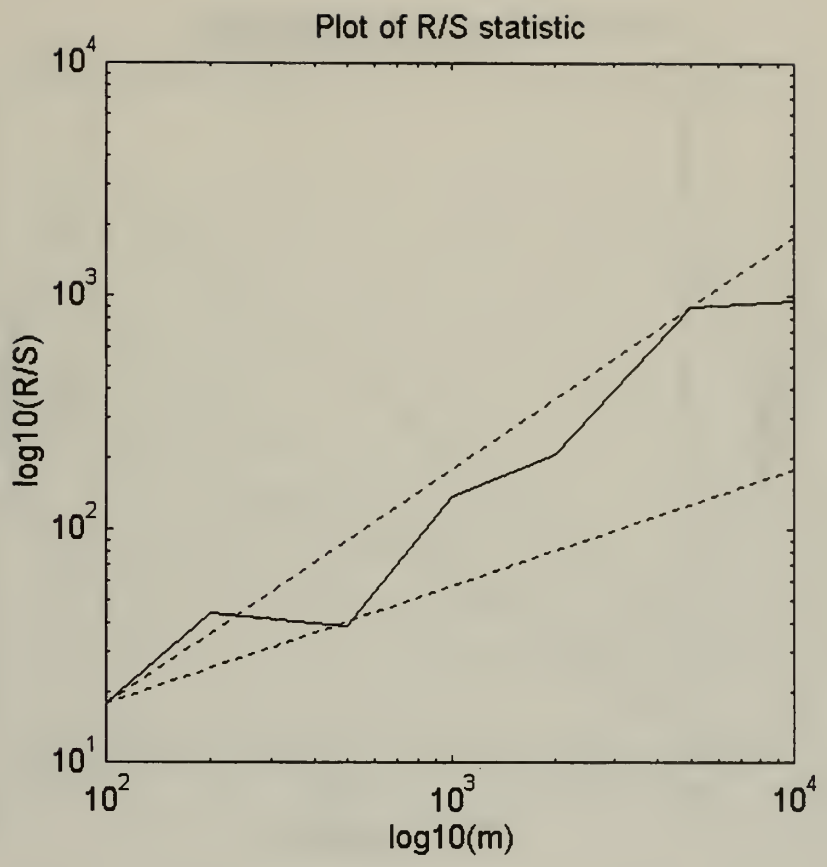


Figure 5.10: Plot of R/S Statistic, Video/Audio Traffic (Measured)

Though a single data point falls outside of the slope region defined by the dashed lines, most of the data points fall within the region illustrating the self-similar nature of the measured traffic.

Figure 5.11 is a plot of the R/S statistic for the simulated traffic, shown in Figure 5.9. This traffic also exhibits characteristics of self-similarity based on the R/S statistic as the asymptotic slope of the R/S statistic data falls between 1/2 and 1 (lower and upper dashed lines, respectively).

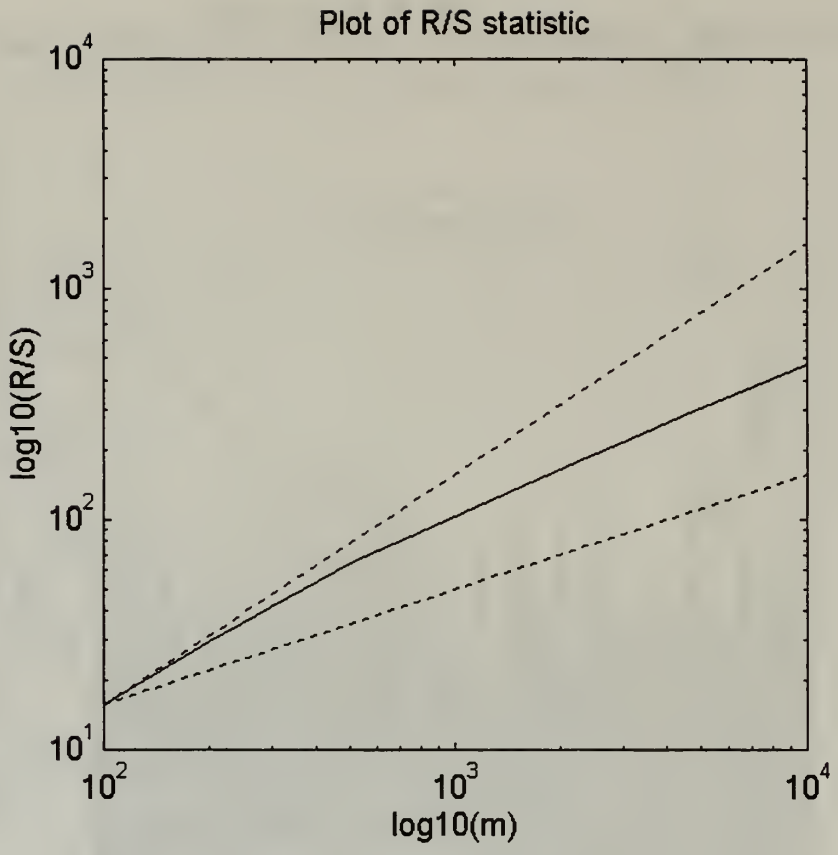


Figure 5.11: Plot of R/S Statistic, Video/Audio Traffic (Simulated)

Figures 5.12 and 5.13 show plots of the variance-time curves for the measured and simulated video/audio traffic, respectively. Again, these plots show characteristics similar to the plots for the video-only traffic case (Figures 5.5 and 5.6). The curve illustrates the self-similar nature of the traffic, in that the asymptotic slope of the curve is approximately -1 or greater. Figure 5.14 shows plots of the IDC curves for the video/audio measured (solid line) and simulated (dashed line) traffic. These plots show that the video/audio traffic is self-similar as indicated by the monitonically increasing slopes of these curves. As in Figure 5.7, the observed gap between the IDC curves for the measured and simulated traffic data is a result of the characteristics of the data (see Figures 5.8 and 5.9), and could be different for different video sequences or simulation runs.

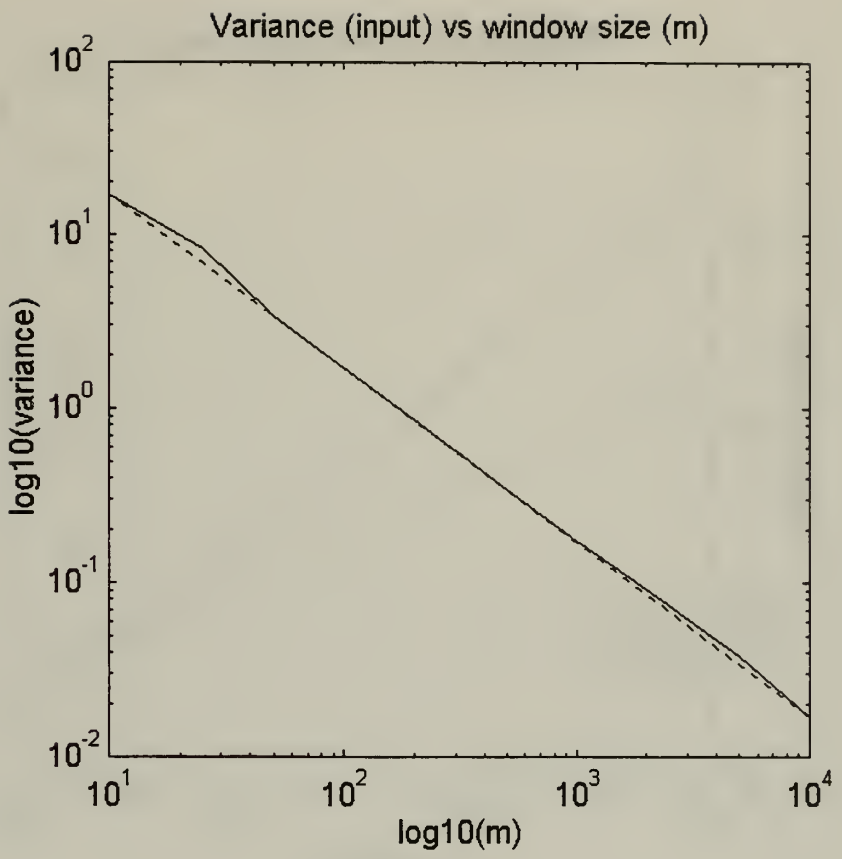


Figure 5.12: Variance-Time Curve, Video/Audio Traffic (Measured)

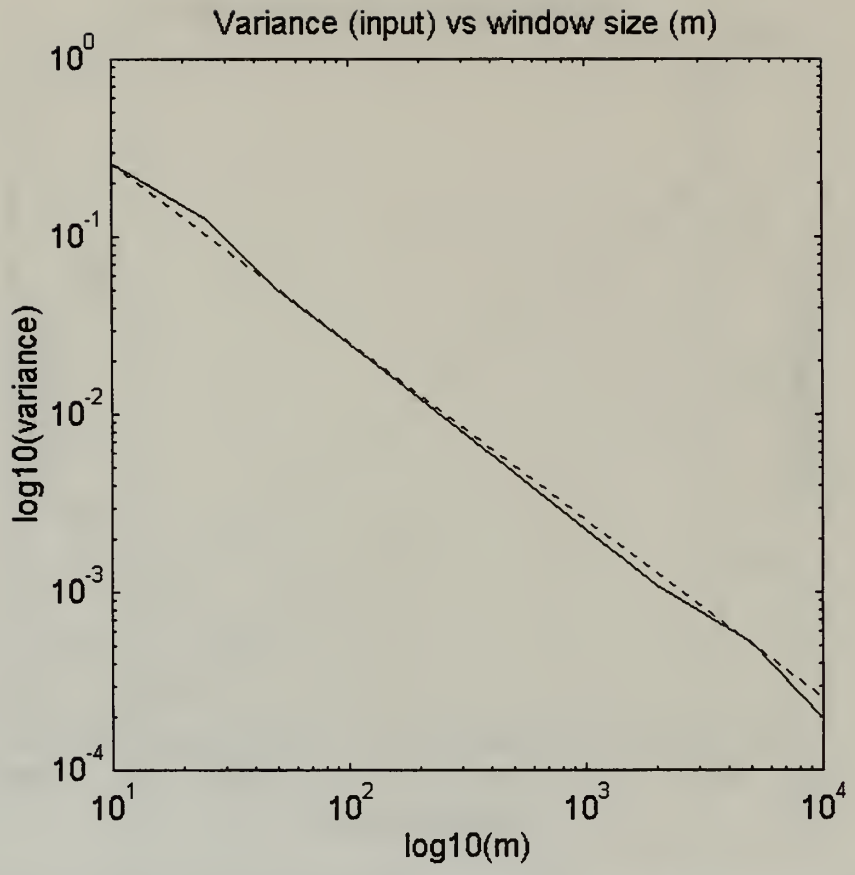


Figure 5.13: Variance-Time Curve for Video/Audio Traffic (Simulated)

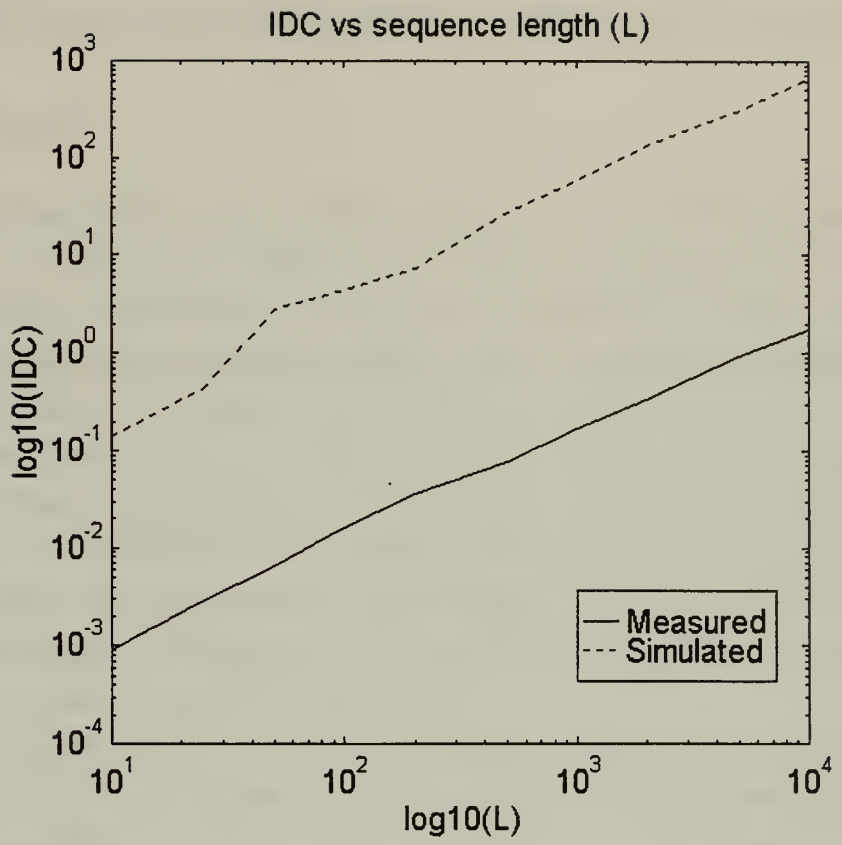


Figure 5.14: IDC Curve for Video/Audio Measured and Simulated Traffic

This chapter has shown, through the graphical representation of three metrics, that the traffic generated by a VTC application and by a VTC simulation model, are self-similar in nature. This conclusion is somewhat restrictive in that the self-similarity is based upon only three metrics (R/S statistic, variance-time curve, and the IDC curve) and a small data sequence (10000 samples) relative to the exhaustive study conducted in [Leland 1994], which used hundreds of thousands of high resolution samples. Perhaps, further study and more metrics are required to fully support the claims of self-similarity made in this chapter.

This concludes the discussion of the results of graphical analysis of traffic captured for this thesis. The next chapter presents the conclusions and recommendations for future work.

VI. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

A. CONCLUSIONS

In this thesis characteristics of VBR traffic generated by a VTC application on an Ethernet LAN were studied in order to validate the self-similar nature of VBR traffic. The traffic studied in this thesis consisted solely of the VBR traffic, whereas others, such as [Leland 1994] have studied Ethernet traffic as a whole, aggregating traffic from many applications (interactive, multimedia, rlogin/telnet, file tranfer, electronic mail, etc.). A Java applet was written in order to facilitate gathering SNMP data from a managed host on the testbed that was participating in a VTC session. Additionally, a Java applet was written in order to simulate the VBR traffic, based on statistical models presented in [Schwartz 1996]. The data gathered by these applets was then analyzed using three metrics (R/S statistic, variance-time curve, and IDC curve) in order to graphically illustrate the self-similar nature of the VBR traffic.

The graphical analysis presented in chapter V illustrated that the VBR traffic produced by both a VTC application and the simulation applets were self-similar in nature. This conclusion supports the assertion that traffic models currently in use are inadequate to describe to the true nature of the traffic found on LANs [Leland 1994]. Based solely on the data gathered and the metrics used in the analysis, the VBR traffic was determined to be self-similar in nature.

Studies such as those conducted in this thesis are extremely important in developing accurate traffic models. As wide area connectivity between LANs is moved to the broadband infrastructure and ATM transport mechanism, research must be conducted in providing admission, access and flow control to all supported traffic types, and especially to VBR traffic, as it is expected to dominate the available bandwidth. VBR traffic produced by a VTC application is one of several types of traffic that will be presented at the access points of the broadband networks. Accordingly, accurate traffic models are required for analysis and study in order to properly configure the broadband networks to support the traffic. For instance, buffers need to be large enough to accommodate the traffic within acceptable loss limits, but not so large as to cause excessive delays in the traffic. This is particularly important when planning to support real-time applications, such as VTC.

Basic concepts of VTC, the testbed and tools used in this thesis, and traffic modeling were presented. A VTC application was used to generate the VBR traffic on the testbed. A Java applet was used to gather SNMP information from a managed host on the testbed in order to derive the necessary data for the thesis. Additionally, a Java applet was used to model and simulate the video-only and aggregate video/audio traffic based on statistical models provided by [Schwartz 1996].

B. RECOMMENDATIONS FOR FUTURE WORK

The tools and metrics used in this thesis applied to study the nature of the traffic gathered from the testbed and the simulation applets. However, further tests should be conducted under a wider range of conditions, using more metrics and a much larger sample set in order show conclusively the self-similar nature of VBR traffic. [Leland 1994] used hundreds of thousands of high resolution samples in order to show that the aggregate traffic gathered from an Ethernet network, without distinguishing traffic types, is self-similar in nature. This thesis has analyzed VBR traffic only, in order to illustrate that this traffic is self-similar in nature.

APPENDIX A: JAVA-SNMP APPLET SOURCE CODE

A. MIBCRUSHER.JAVA

```
/************************************************************************
* author:  H. Carvey
* file:    MibCrusher.java
* usage:   appletviewer MibCrusher.html
*
* Note:    Due to security, this applet will not work in Netscape, as
*          you may not write files to the hard drive.  Appletviewer
*          must be used.
*
*          This file provides the graphical front-end and exception
*          handling for the applet.
*
* Environment: WindowsNTServer 3.51, Symantec Cafe 1.5
* Project files:  PollThread.java, snmpGet.java
*
*
* 26 Oct 96
************************************************************************/
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import Snmp.*;

public class MibCrusher extends Applet {

   Button Start, Stop, About;
   TextField host,file,pollInt;
   int poll_int;
   TextArea messages;
   String def_log = "test1.dat";
   String hostIP = null;
   String logfile = null;

// Thread
   PollThread pollThread;

// Fonts
   Font titleFont = new Font("Helvetica", Font.BOLD + Font.ITALIC, 24);
```

```
// init method
   public void init() {

// containers for all components
      Panel panel1, panel2, buttonpanel;

// Layout manager for each panel
      GridBagLayout gridbag = new GridBagLayout();

// components
      Start = new Button("Start Polling");
      Stop = new Button("Stop Polling");
      About = new Button("About");

      file = new TextField(15);
      file.setEditable(true);
      file.setText(def_log);


      host = new TextField(15);
      host.setEditable(true);
      host.setText("131.120.122.101");

      pollInt = new TextField(15);
      pollInt.setEditable(true);
      pollInt.setText("1");
      messages = new TextArea(12,40);
      messages.setEditable(false);

// panel1
      panel1 = new Panel();
      panel1.setLayout(gridbag);
      constrain(panel1, new Label("IP Address for host:"),0,0,1,1);
      constrain(panel1, host,0,1,1,1);
      constrain(panel1, new Label("LogFile for host:"),0,2,1,1);
      constrain(panel1, file,0,3,1,1);
      constrain(panel1, new Label("Polling Interval (sec): "),0,4,1,1);
      constrain(panel1, pollInt,0,5,1,1);


// panel2
      panel2 = new Panel();
      panel2.setLayout(gridbag);
      constrain(panel2, new Label("Messages: "),0,0,1,1);
      constrain(panel2, messages, 0,1,1,3, GridBagConstraints.HORIZONTAL,
```

```
            GridBagConstraints.NORTH,1.0,0.0,0,0,0,10);

// buttonpanel
      buttonpanel = new Panel();
      buttonpanel.setLayout(gridbag);
      constrain(buttonpanel,Start,0,0,1,1,GridBagConstraints.NONE,
            GridBagConstraints.CENTER,0.3,0.0,0,0,10,0);
      constrain(buttonpanel,Stop,1,0,1,1,GridBagConstraints.NONE,
            GridBagConstraints.CENTER,0.3,0.0,0,0,10,0);
      constrain(buttonpanel,About,2,0,1,1,GridBagConstraints.NONE,
            GridBagConstraints.CENTER,0.3,0.0,0,0,10,0);

// Label
      Label name = new Label("MIB Crusher, ver 1.0", Label.CENTER);
      name.setFont(titleFont);

// arrange panels in main applet panel
      this.setLayout(new BorderLayout());
      this.add("North",name);
      this.add("West",panel1);
      this.add("Center",panel2);
      this.add("South",buttonpanel);

   }// init()

// main constrain methods for GridBagLayout
   public void constrain(Container container, Component component,
                  int grid_x, int grid_y, int grid_width, int grid_height,
                  int fill, int anchor, double weight_x, double weight_y,
                  int top, int left, int bottom, int right)
   {
      GridBagConstraints c = new GridBagConstraints();
      c.gridx = grid_x; c.gridy = grid_y;
      c.gridwidth = grid_width; c.gridheight = grid_height;
      c.fill = fill; c.anchor = anchor;
      c.weightx = weight_x; c.weighty = weight_y;
      if (top+bottom+left+right > 0)
         c.insets = new Insets(top,left,bottom,right);

      ((GridBagLayout)container.getLayout()).setConstraints(component,c);
      container.add(component);
   }

   public void constrain(Container container,Component component,
                  int grid_x, int grid_y, int grid_width, int grid_height)
```

```
    {
      constrain(container,component,grid_x,grid_y,grid_width,grid_height,
        GridBagConstraints.NONE,GridBagConstraints.NORTHWEST,0.0,0.0,0,10,0,10);
    }

    public void constrain(Container container, Component component,
                  int grid_x, int grid_y, int grid_width, int grid_height,
                  int top, int left, int bottom, int right)
    {
      constrain(container,component,grid_x,grid_y,grid_width,grid_height,
        GridBagConstraints.NONE,GridBagConstraints.NORTHWEST,0.0,0.0,top,
        left,bottom,right);
    }

// event handler method
    public boolean handleEvent(Event evt) {

      if (evt.id == Event.ACTION_EVENT && evt.target == Start) {

        StartPolling();
        return true;
      }
      else if (evt.id == Event.ACTION_EVENT && evt.target == Stop) {
        StopPolling();

        return true;
      }
      else if (evt.id == Event.ACTION_EVENT && evt.target == About) {
        messages.appendText("\nMIB Crusher\n");
        messages.appendText("by:  Harlan Carvey\n");
        messages.appendText("Special Thanks to: \n");
        messages.appendText(" - Sun Microsystems, for their way\n");
        messages.appendText("   kewl programming language, Java\n");
        messages.appendText(" - Advent Network Mgmt, for their most\n");
        messages.appendText("   excellent SNMP packa!e for Java\n");

        return true;
      }

      return super.handleEvent(evt);
    }// event handler

 // StartPolling method
    private void StartPolling() {
```

```
        logfile = file.getText();
        hostIP = host.getText();

        int poll_int = Integer.parseInt(pollInt.getText());
        pollThread = new PollThread(messages,hostIP,logfile,poll_int);
        pollThread.start();
        messages.appendText("Polling Thread started...\n");

    }// StartPolling method


// StopPolling method

    private void StopPolling() {

        pollThread.CloseLogFile();
        pollThread.stop();
        messages.appendText("Polling stopped.\n");

    } // StopPolling method

}// class MibCruiser
```

B. POLLTHREAD.JAVA

```
/**************************************************************************
* author:  H. Carvey
* file:    PollThread.java
* File is the thread for sending get-requests
*         Applet sends multiple OIDs in each get-request to the
*         host, at user-defined interval, and prints results to
*         user-defined log file.  Defaults provided.
* Constructor: PollThread(TextArea, String[], int)
* Environment:  WindowsNTServer 3.51, Symantec Cafe 1.5
* Proj files:  MibCrusher.java, snmpGet.java
*
* 26 Oct 96
**************************************************************************/
import java.io.*;
import java.awt.*;
import java.util.*;
import Snmp.*;
import snmpGet;

public class PollThread extends Thread {
```

```
// OIDs used in original implementation of the application
// String oid[] = {"2.2.1.10.1","2.2.1.16.1","4.3.0","4.10.0"};
   String oid[] = {"2.2.1.10.2","2.2.1.16.2"};
   File Log;
   FileOutputStream os;
   PrintStream ps;

   String host;
   String logFile;
   snmpGet snmp_get;
   TextArea output;
   int pollcount = 1;
   int pause;

   SnmpAPI api = new SnmpAPI();
   SnmpSession session;

// constructor
   public PollThread(TextArea messages, String Host, String logfile, int delay){
      output = messa!es;
      host = Host;
      logFile = logfile;
      pause = delay;
   }//constructor

   public void run() {
      api.start();

// instantiate a new session
      session = new SnmpSession(api);
      session.peername = host;
      session.community = "public";
      session.remote_port = 161;
      session.retries = 0;
      session.timeout = 10000;

      try {
         session.open();
      }
      catch(SnmpException e) {
         System.out.println("Cannot open session.");
      }

      OpenLogFiles();
```

```
        long start = System.currentTimeMillis();

        while(true){

            ps.print((System.currentTimeMillis() - start) + " ");

// send PDU's to host
            snmp_get = new snmpGet(api,session,oid,host,ps);
            output.setText("PDU sent to host: " + host + "\n");

            output.appendText("Poll Count: " + pollcount + "\n");
            pollcount++;
            output.appendText("Pause for " + pause + " sec delay.\n");

            System.gc();

            try {
               this.sleep(pause*1000);
            }
            catch (InterruptedException e) {
               output.appendText("Thread interrupted\n");
            }

        }// while
    }//run

// OpenLogFiles method
    private void OpenLogFiles() {
        Date today = new Date();

            Log = new File(logFile);
            try {
               os = new FileOutputStream(Log);
            }
            catch (IOException e) {
               System.out.println("Could not open file os");
            }

            ps = new PrintStream(os);

    }//OpenLogFile method

// CloseLogFile method
        public void CloseLogFile() {
```

```
        output.appendText("Closing Log Files\n");
        try {
           os.close();
        }
        catch (IOException e){
           output.appendText("IOException in method CloseLogFile.\n");
        }
     }//CloseLogFile

}//class
```

C. SNMPGET.JAVA

```
/*********************************************************************
* author:  H. Carvey
* file:    snmpGet.java
* usage:   snmpGet(SnmpAPI, String[], String, PrintStream)
*          Prints variable bindings to PrintStream
* Environment:  WindowsNTServer 3.51, Symantec Cafe 1.5
* Project files: MibCrusher.java, PollThread.java
*
* 26 Oct 96
*********************************************************************/
import java.io.*;
import java.util.*;
import Snmp.*;

public class snmpGet {

// constructor
   public snmpGet(SnmpAPI api, SnmpSession session, String remArgs[], String
host,PrintStream outFile){

      SnmpPDU pdu = new SnmpPDU(api);
      pdu.command = api.GET_REQ_MSG;

      int x = remArgs.length;

      for (int i=0;i<x;i++) {
         SnmpOID oid = new SnmpOID(remArgs[i],api);
         if (oid.toValue() == null) System.err.println("Invalid OID Arg: " + remArgs[i]);
         else pdu.addNull(oid);
      }

      try {
```

```
        pdu = session.syncSend(pdu);
      }
      catch (SnmpException e) {
        System.err.println("Sending PDU: " + e.getMessage());
      }


      if (pdu == null) {
        System.out.println("Request timed out to: " + session.peername);
        System.exit(1);
      }

      if (pdu.errstat != 0)
        System.out.println("Error Indication: " + SnmpException.exceptionString ((byte)
          pdu.errstat) + "\nErrindex: " + pdu.errindex);

      String varbinds = pdu.printVarBinds();
      StringTokenizer t = new StringTokenizer(varbinds);
      int tokens = t.countTokens();
      String value[] = new String[tokens];

      for (int i = 0;i <= tokens-1;i++){
        value[i] = t.nextToken();
      }

      outFile.println(value[4] + " " + value[9]);

    } // constructor
}// class file
```

D. MIBCRUSHER.HTML

```
<!--MibCrusher.html-->
<HTML>
<HEAD>
<TITLE>MIBCrusher</TITLE>
</HEAD>
<BODY>
<HR>
<APPLET CODE=MibCrusher.class WIDTH=700 HEIGHT=450>

</APPLET>

</BODY>
</HTML>
```

APPENDIX B: JAVA SIMULATION APPLET SOURCE CODE

A. SIMTEST.JAVA

```
/*************************************************************
* H. Carvey
* Simulation applet for thesis
* Written as an applet but will only run in the
* appletviewer; will not run in a browser due to security
*
* <!--SimTest.html-->
* <HTML>
* <HEAD>
* <TITLE>SimTest</TITLE>
* </HEAD>
* <BODY>
* <HR>
* <APPLET CODE=SimTest.class WIDTH=500 HEIGHT=250>
* </APPLET>
* </BODY>
* </HTML>
*
* Command:  appletviewer SimTest.html
*************************************************************/

/*************************************************************
*  SimTest.java
*  GUI and exception handler for simulation
*************************************************************/
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.util.Date;
import java.util.Random;

public class SimTest extends Applet {

   Button Start, Stop, About, Close;
   TextField file;
   TextField messages;
   SimThread s_thread;

   public void init() {

      Panel panel1, panel2, buttonpanel;
```

```
        GridBagLayout gridbag = new GridBagLayout();

        Start = new Button("Start");
        Stop = new Button("Stop");
        About = new Button("About");
        Close = new Button("Close");

        file = new TextField(15);
        file.setEditable(true);
        file.setBackground(Color.white);
        file.setText("sim1.dat");

        messages = new TextField(15);
        messages.setEditable(false);
        messages.setBackground(Color.lightGray);

// panel1
        panel1 = new Panel();
        panel1.setBackground(Color.lightGray);
        panel1.setLayout(gridbag);
        constrain(panel1, new Label("LogFile:"),0,2,1,1);
        constrain(panel1, file,0,3,1,1);

// panel2
        panel2 = new Panel();
        panel2.setBackground(Color.lightGray);
        panel2.setLayout(gridbag);
        constrain(panel2, new Label("Messages: "),0,0,1,1);
        constrain(panel2, messages, 0,1,1,3, GridBagConstraints.HORIZONTAL,
                GridBagConstraints.NORTH,1.0,0.0,0,0,0,10);

// buttonpanel
        buttonpanel = new Panel();
        buttonpanel.setBackground(Color.lightGray);
        buttonpanel.add(new Label("Simulation: "));
        buttonpanel.add(Start);
        buttonpanel.add(Stop);
        buttonpanel.add(About);
        buttonpanel.add(Close);
// Label
        Label name = new Label("Video Source Simulation", Label.CENTER);
        name.setFont(new Font("Helvetica", Font.BOLD + Font.ITALIC, 32));

// arrange panels in main applet panel
```

```
        this.setLayout(new BorderLayout());
        this.setBackground(Color.lightGray);
        this.add("North",name);
        this.add("West",panel1);
        this.add("East",panel2);
        this.add("South",buttonpanel);

    }// init()

// main constrain methods for GridBagLayout
    public void constrain(Container container, Component component,
                    int grid_x, int grid_y, int grid_width, int grid_height,
                    int fill, int anchor, double weight_x, double weight_y,
                    int top, int left, int bottom, int right)
    {
        GridBagConstraints c = new GridBagConstraints();
        c.gridx = grid_x; c.gridy = grid_y;
        c.gridwidth = grid_width; c.gridheight = grid_height;
        c.fill = fill; c.anchor = anchor;
        c.weightx = weight_x; c.weighty = weight_y;
        if (top+bottom+left+right > 0)
            c.insets = new Insets(top,left,bottom,right);

        ((GridBagLayout)container.getLayout()).setConstraints(component,c);
        container.add(component);
    }

    public void constrain(Container container,Component component,
                    int grid_x, int grid_y, int grid_width, int grid_height)
    {
        constrain(container,component,grid_x,grid_y,grid_width,grid_height,

GridBagConstraints.NONE,GridBagConstraints.NORTHWEST,0.0,0.0,0,10,0,10);
    }

    public void constrain(Container container, Component component,
                    int grid_x, int grid_y, int grid_width, int grid_height,
                    int top, int left, int bottom, int right)
    {
        constrain(container,component,grid_x,grid_y,grid_width,grid_height,
            GridBagConstraints.NONE,GridBagConstraints.NORTHWEST,0.0,0.0,top,
            left,bottom,right);
    }

// event handler method
```

```
    public boolean handleEvent(Event evt) {

        if (evt.id == Event.ACTION_EVENT && evt.target == Start) {
            showStatus("Simulation started...");
            s_thread = new SimThread(file.getText(), messages);
            s_thread.openLogFile();
            s_thread.start();
            return true;

        }
        else if (evt.id == Event.ACTION_EVENT && evt.target == Stop) {
            s_thread.closeLogFile();
            s_thread.stop();
            showStatus("Simulation stopped.");
            return true;
        }
        else if (evt.id == Event.ACTION_EVENT && evt.target == Close) {
            System.exit(0);
            return true;
        }
        else if (evt.id == Event.ACTION_EVENT && evt.target == About) {
            showStatus("About");
            messages.setText("SimTest");
            return true;
        }

        return super.handleEvent(evt);
    }// event handler

}// class SimTest


/*****************************************************************
* SimThread.java file
* Handles the actual computations and transitions between
* states of the FSM
*****************************************************************/

class SimThread extends Thread {

    private double A = 0.128416;
    private double ALPHA = 0.78623;
    private double BETA = 3.1138;
    private double FSM_UP[] = new double[21];
    private double FSM_DOWN[] = new double[21];
```

```
   private int STATE;
   private long TIMESTAMP;
   private long START;
   Node newNode;
   private int i;
   String logfile;
   File Log;
   FileOutputStream os;
   PrintStream ps;
   TextField status;
   int counter;

   public SimThread(String filename, TextField ta) {

/*
*  Initialize variables and get things ready for the
*  simulation to run unabated
*/
      logfile = filename;
      newNode = new Node();
      STATE = 0;
      setTransitionArrays();
      START = System.currentTimeMillis();
      status = ta;
   }

   public void run(){
      counter = 1;
      while(true) {

/*
*  Once nextNode() is returned, then decide the new STATE
*  of the FSM
*/
         i = newNode.nextNode(FSM_UP[STATE] , FSM_DOWN[STATE]);
         if (i == 1) STATE++;
         else if (i == -1) STATE--;
         else STATE = STATE;

/*
* Need to limit the number of STATEs to the range of
* 0 - 20
*
* Conversion factor for A bits/pixel to bytes/slot transmitted
```

```
* is STATE * 924.9
* This gives the number of bytes transmitted per 100 msec time-
* slot
*/
        TIMESTAMP = System.currentTimeMillis() - START;

        if (STATE <= 0) {
           ps.println(TIMESTAMP + "\t" + (STATE * 924.9));
           STATE = 1;
        }

        if (STATE >= 20) {
           ps.println(TIMESTAMP + "\t" + (STATE * 924.9));
           STATE = 19;
        }

        ps.println(TIMESTAMP + "\t" + (STATE * 924.9));

        try {
           Thread.sleep(100);
        }
        catch(InterruptedException e) {

        }
        counter++;
        status.setText("Event: " + counter);

     }
  }

  public void openLogFile() {

        Log = new File(logfile);
        try {
           os = new FileOutputStream(Log);
        }
        catch (IOException e) {
           System.out.println("Could not open file outputstream");
        }

        ps = new PrintStream(os);
  }

  public void closeLogFile() {
```

```
        try {
            os.close();
        }
        catch (IOException e) {
            System.out.println("Could not close logfile");
        }
    }

    private void setTransitionArrays() {

        for (int i=0;i<21;i++) {
            FSM_DOWN[i] = i * BETA;
            FSM_UP[i] = (20 - i) * ALPHA;
        }

    }

}

/*
* Class to represent the Nodes of the FSM
*/
class Node {

    double X,Y;
    Random r;

    public Node() {

        r = new Random();
    }

    public int nextNode(double UP, double DOWN) {

        X = ((-1)/UP) * Math.log(r.nextDouble());
        Y = ((-1)/DOWN) * Math.log(r.nextDouble());
        if (X <= Y) return 1;
        else if (X > Y) return -1;
        else return 0;
    }
}
```

B. SIMTEST2.JAVA

```
/*************************************************************
* H. Carvey
* Video/Audio Simulation applet for thesis
* Written as an applet but will only run in the
* appletviewer; will not run in a browser due to security
*
* <!--SimTest2.html-->
* <HTML>
* <HEAD>
* <TITLE>SimTest2</TITLE>
* </HEAD>
* <BODY>
* <HR>
* <APPLET CODE=SimTest2.class WIDTH=500 HEIGHT=250>
* </APPLET>
* </BODY>
* </HTML>
*
* Command:  appletviewer SimTest2.html
*************************************************************/

/*************************************************************
*  SimTest2.java
*  GUI and exception handler for video/audio simulation
*************************************************************/
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.util.Date;
import java.util.Random;

public class SimTest2 extends Applet {

   Button Start, Stop, About, Close;
   TextField file;
   TextField messages;
   SimThread2 s_thread;

   public void init() {

      Panel panel1, panel2, buttonpanel;

      GridBagLayout gridbag = new GridBagLayout();
```

```
        Start = new Button("Start");
        Stop = new Button("Stop");
        About = new Button("About");
        Close = new Button("Close");

        file = new TextField(15);
        file.setEditable(true);
        file.setBackground(Color.white);
        file.setText("sim2.dat");

        messages = new TextField(15);
        messages.setEditable(false);
        messages.setBackground(Color.lightGray);

// panel1
        panel1 = new Panel();
        panel1.setBackground(Color.lightGray);
        panel1.setLayout(gridbag);
        constrain(panel1, new Label("LogFile:"),0,2,1,1);
        constrain(panel1, file,0,3,1,1);

// panel2
        panel2 = new Panel();
        panel2.setBackground(Color.lightGray);
        panel2.setLayout(gridbag);
        constrain(panel2, new Label("Messages: "),0,0,1,1);
        constrain(panel2, messages, 0,1,1,3, GridBagConstraints.HORIZONTAL,
                GridBagConstraints.NORTH,1.0,0.0,0,0,0,10);

// buttonpanel
        buttonpanel = new Panel();
        buttonpanel.setBackground(Color.lightGray);
        buttonpanel.add(new Label("Simulation: "));
        buttonpanel.add(Start);
        buttonpanel.add(Stop);
        buttonpanel.add(About);
        buttonpanel.add(Close);
// Label
        Label name = new Label("Video/Audio Source Simulation", Label.CENTER);
        name.setFont(new Font("Helvetica", Font.BOLD + Font.ITALIC, 32));

// arrange panels in main applet panel
        this.setLayout(new BorderLayout());
        this.setBackground(Color.lightGray);
```

```
        this.add("North",name);
        this.add("West",panel1);
        this.add("East",panel2);
        this.add("South",buttonpanel);

    }// init()

// main constrain methods for GridBagLayout
    public void constrain(Container container, Component component,
                    int grid_x, int grid_y, int grid_width, int grid_height,
                    int fill, int anchor, double weight_x, double weight_y,
                    int top, int left, int bottom, int right)
    {
        GridBagConstraints c = new GridBagConstraints();
        c.gridx = grid_x; c.gridy = grid_y;
        c.gridwidth = grid_width; c.gridheight = grid_height;
        c.fill = fill; c.anchor = anchor;
        c.weightx = weight_x; c.weighty = weight_y;
        if (top+bottom+left+right > 0)
            c.insets = new Insets(top,left,bottom,right);

        ((GridBagLayout)container.getLayout()).setConstraints(component,c);
        container.add(component);
    }

    public void constrain(Container container,Component component,
                    int grid_x, int grid_y, int grid_width, int grid_height)
    {
        constrain(container,component,grid_x,grid_y,grid_width,grid_height,

GridBagConstraints.NONE,GridBagConstraints.NORTHWEST,0.0,0.0,0,10,0,10);
    }

    public void constrain(Container container, Component component,
                    int grid_x, int grid_y, int grid_width, int grid_height,
                    int top, int left, int bottom, int right)
    {
        constrain(container,component,grid_x,grid_y,grid_width,grid_height,
            GridBagConstraints.NONE,GridBagConstraints.NORTHWEST,0.0,0.0,top,
            left,bottom,right);
    }

// event handler method
    public boolean handleEvent(Event evt) {
```

```
        if (evt.id == Event.ACTION_EVENT && evt.target == Start) {
            showStatus("Simulation started...");
            s_thread = new SimThread2(file.getText(), messages);
            s_thread.openLogFile();
            s_thread.start();
            return true;

        }
        else if (evt.id == Event.ACTION_EVENT && evt.target == Stop) {
            s_thread.closeLogFile();
            s_thread.stop();
            showStatus("Simulation stopped.");
            return true;
        }
        else if (evt.id == Event.ACTION_EVENT && evt.target == Close) {
            System.exit(0);
            return true;
        }
        else if (evt.id == Event.ACTION_EVENT && evt.target == About) {
            showStatus("About");
            messages.setText("SimTest");
            return true;
        }

        return super.handleEvent(evt);
    }// event handler

}// class SimTest


/*****************************************************************
* SimThread2.java
* Handles the actual computations and transitions between
* states of the FSM
*****************************************************************/

class SimThread2 extends Thread {

    private double A = 0.128416;
    private double ALPHA = 0.78623;
    private double BETA = 3.1138;
    private double FSM_UP[] = new double[21];
    private double FSM_DOWN[] = new double[21];

    private int STATE;
```

```
   private long TIMESTAMP;
   private long START;
   Node newNode;
   private int i;
   String logfile;
   File Log;
   FileOutputStream os;
   PrintStream ps;
   TextField status;
   int counter;

   AudioSim a_thread;

   public SimThread2(String filename, TextField ta) {
/*
*  Initialize variables and get things ready for the
*  simulation to run unabated
*/
      logfile = filename;
      newNode = new Node();
      STATE = 0;
      setTransitionArrays();
      START = System.currentTimeMillis();
      status = ta;
   }

   public void run(){

      a_thread = new AudioSim();
      a_thread.start();

      counter = 1;
      while(true) {

/*
*   Once nextNode() is returned, then decide the new STATE
*   of the FSM
*/
         i = newNode.nextNode(FSM_UP[STATE] , FSM_DOWN[STATE]);
         if (i == 1) STATE++;
         else if (i == -1) STATE--;
         else STATE = STATE;

/*
*   Need to limit the number of STATEs to the range of
```

```
*  0 - 20
*
*  Conversion factor for A bits/pixel to bytes/slot transmitted
*  is STATE * 924.9
*  This gives the number of bytes transmitted per 100 msec time-
*  slot
*
*  Must add in the number of bytes produced by the audio component
*/
        TIMESTAMP = System.currentTimeMillis() - START;

        if (STATE <= 0) {
           ps.println(TIMESTAMP + "\t" + ((STATE * (double)924.9) +
              (a_thread.getXmitBytes()/10)));
           STATE = 1;
        }

        if (STATE >= 20) {
           ps.println(TIMESTAMP + "\t" + ((STATE * (double)924.9) +
              (a_thread.getXmitBytes()/10)));
           STATE = 19;
        }

        ps.println(TIMESTAMP + "\t" + ((STATE * (double)924.9) +
              (a_thread.getXmitBytes()/10)));

        try {
           Thread.sleep(110);
        }
        catch(InterruptedException e) {

        }
        counter++;
        status.setText("Event: " + counter);

     }
  }

  public void openLogFile() {

        Log = new File(logfile);
        try {
           os = new FileOutputStream(Log);
        }
        catch (IOException e) {
```

```
            System.out.println("Could not open file outputstream");
        }

        ps = new PrintStream(os);
    }

    public void closeLogFile() {
/*
*  Must halt the audio simulation thread, then close the logfile
*
*/
        a_thread.stop();

        try {
            os.close();
        }
        catch (IOException e) {
            System.out.println("Could not close logfile");
        }
    }

    private void setTransitionArrays() {

        for (int i=0;i<21;i++) {
            FSM_DOWN[i] = i * BETA;
            FSM_UP[i] = (20 - i) * ALPHA;
        }

    }

}

/* Class to represent the Nodes of the FSM
*/
class Node {

    double X,Y;
    Random r;

    public Node() {

        r = new Random();
    }

    public int nextNode(double UP, double DOWN) {
```

```
        X = ((-1)/UP) * Math.log(r.nextDouble());
        Y = ((-1)/DOWN) * Math.log(r.nextDouble());
        if (X <= Y) return 1;
        else if (X > Y) return -1;
        else return 0;
      }
}

/*
* AudioNode files for SimTest2
*
*/

class AudioNode {

   double expValue;
   Random r;

   public AudioNode() {
      expValue = 0;
      r = new Random();
   }

   public AudioNode(double value) {
      expValue = value;
      r = new Random();
   }

   public double time() {
      double X = ((-1)/expValue) * Math.log(r.nextDouble());
      return X;
   }
}
```

```
/*
* Class to provide a separate thread for the audio
* simulation component
*/
class AudioSim extends Thread {

   AudioNode onNode, offNode;
   double xmitBytes, wait;

   public AudioSim() {
      onNode = new AudioNode(2.5);
      offNode = new AudioNode(1.67);
      xmitBytes = 0.0;
   }

   public void run() {

      while(true) {
         wait = offNode.time();
         try {
            Thread.sleep((long)wait);
         }
         catch(InterruptedException e) {
            System.out.println("Thread sleep interrupted");
         }
         wait = onNode.time();
         try {
            Thread.sleep((long)wait);
         }
         catch(InterruptedException e) {
            System.out.println("Thread sleep interrupted");
         }
         xmitBytes += wait * 8.0;
      }
   }

   public double getXmitBytes() {
      double getBytes = xmitBytes;
      xmitBytes = 0.0;
      return getBytes;
   }
}
```

APPENDIX C: MATLAB SOURCE CODE

A. TRAFFIC.M

```
%*****************************************************
% Harlan Carvey
%
% file for displaying data for thesis
% - produces graph of traffic in KBps per
% unit time
%*****************************************************

clear all;
load v1114.dat;

[M N] = size(v1114);

timer = v1114(:,1);
i_oct = v1114(:,2);

clear v1114;

for i = 2:M;
  time(i-1) = timer(i) - timer(i-1);
  input(i-1) = i_oct(i) - i_oct(i-1);
  h_time(i-1) = timer(i) - timer(1);
end;

i_kbps = (input .* 1000) ./ (time .* 1024);

% display KBps on y-axis, units of time on x-axis
% x-axis represents the total number of msec of the
% test
% use bar vice plot to represent the data

bar(h_time,i_kbps,'k-'),grid;
axis([0 max(h_time) 0 max(i_kbps)+0.5]);
title('Input Traffic');
xlabel('msec');ylabel('KBps');
```

B. RS.M

```
%*********************************************
% Harlan Carvey
%
% file for computing RS statistic
% - produces R/S statistic plot
%*********************************************

clear all;
load v1114.dat;

[M N] = size(v1114);

timer = v1114(:,1);
i_oct = v1114(:,2);

clear v1114;

for i = 2:M;
  time(i-1) = timer(i) - timer(i-1);
  input(i-1) = i_oct(i) - i_oct(i-1);
end;

i_kbps = (input .* 1000) ./ (time .* 1024);

i = i_kbps(1:10000);
x1 = i;

x = [100 200 500 1000 2000 5000 10000];

w = 5000;
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
x2 = s./t;

w = 2000;
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
```

```
x3 = s./t;

w = 1000;
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
x4 = s./t;

w = 500;
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
x5 = s./t;

w = 200;
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
x6 = s./t;

w = 100;
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
x7 = s./t;


% R/S statistic section

i = x7;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(1) = (max(W) - min(W))/s;
```

```
i = x6;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(2) = (max(W) - min(W))/s;

i = x5;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(3) = (max(W) - min(W))/s;

i = x4;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(4) = (max(W) - min(W))/s;

i = x3;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(5) = (max(W) - min(W))/s;

i = x2;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(6) = (max(W) - min(W))/s;
```

```
i = x1;
m = mean(i);
s = sqrt(cov(i));
l = length(i);
for k = 1:l;
  W(k) = sum(i(1:k)) - (k*m);
end;
RS(7) = (max(W) - min(W))/s;

loglog(x,RS,'k',x,RS,'ko'),grid;
title('Plot of R/S statistic');
xlabel('log10(m)');ylabel('log10(R/S)');
```

C. VAR.M

```
%************************************************
% Harlan Carvey
%
% file for displaying data for thesis
% - produces variance-time curve for data
%************************************************

clear all;
load va1114.dat;

[M N] = size(va1114);

timer = va1114(:,1);
i_oct = va1114(:,2);

clear va1114;

for i = 2:M;
  time(i-1) = timer(i) - timer(i-1);
  input(i-1) = i_oct(i) - i_oct(i-1);
end;

i_kbps = (input .* 1000) ./ (time .* 1024);

% determine variance of original sequence and
% sequences produced using window sizes represented
% in vector x

i = i_kbps(1:10000);
```

```
ic(1) = cov(i);

x = [10 25 50 100 200 500 1000 2000 5000 10000];

w = x(9);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(2) = cov(s);

w = x(8);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(3) = cov(s);

w = x(7);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(4) = cov(s);

w = x(6);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(5) = cov(s);

w = x(5);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
```

```
t = length(i)/w;
s = s./t;
ic(6) = cov(s);

w = x(4);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(7) = cov(s);

w = x(3);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(8) = cov(s);

w = x(2);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(9) = cov(s);

w = x(1);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
ic(10) = cov(s);

loglog(x,ic,'k',x,ic,'k+'),grid;
title('Variance (input) vs window size (m)');
xlabel('log10(m)');ylabel('log10(variance)');
```

D. IDCM.M

```
%************************************************
% Harlan Carvey
%
% file for displaying data for thesis
% - produces IDC curves for measured and
%   simulated data (video/audio data)
%************************************************
clear all;
load va1114.dat;

[M N] = size(va1114);
timer = va1114(:,1);
i_oct = va1114(:,2);
clear va1114;

for i = 2:M;
  time(i-1) = timer(i) - timer(i-1);
  input(i-1) = i_oct(i) - i_oct(i-1);
end;

i_kbps = (input .* 1000) ./ (time .* 1024);

% determine variance of original sequence and
% sequences produced using window sizes represented
% in vector x

i = i_kbps(1:10000);
idc(10) = cov(i)/mean(i);

x = [10 25 50 100 200 500 1000 2000 5000 10000];

w = x(9);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(9) = cov(s)/mean(s);

w = x(8);
s = zeros([1,w]);
for k = 1:w:length(i);
```

```
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(8) = cov(s)/mean(s);

w = x(7);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(7) = cov(s)/mean(s);

w = x(6);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(6) = cov(s)/mean(s);

w = x(5);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(5) = cov(s)/mean(s);

w = x(4);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(4) = cov(s)/mean(s);

w = x(3);
s = zeros([1,w]);
for k = 1:w:length(i);
```

```
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(3) = cov(s)/mean(s);

w = x(2);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(2) = cov(s)/mean(s);

w = x(1);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(1) = cov(s)/mean(s);

loglog(x,idc,'k');
title('IDC vs sequence length (L)');
xlabel('log10(L)');ylabel('log10(IDC)');
hold on;

clear all;
load sim2.dat;

time = sim2(1:10000,1);
input = sim2(1:10000,2);

clear sim2;

i = input';
idc(10) = cov(i)/mean(i);

x = [10 25 50 100 200 500 1000 2000 5000 10000];

w = x(9);
s = zeros([1,w]);
for k = 1:w:length(i);
```

```
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(9) = cov(s)/mean(s);

w = x(8);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(8) = cov(s)/mean(s);

w = x(7);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(7) = cov(s)/mean(s);

w = x(6);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(6) = cov(s)/mean(s);

w = x(5);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(5) = cov(s)/mean(s);

w = x(4);
s = zeros([1,w]);
for k = 1:w:length(i);
```

```
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(4) = cov(s)/mean(s);

w = x(3);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(3) = cov(s)/mean(s);

w = x(2);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(2) = cov(s)/mean(s);

w = x(1);
s = zeros([1,w]);
for k = 1:w:length(i);
  s = s + i(k:k+w-1);
end;
t = length(i)/w;
s = s./t;
idc(1) = cov(s)/mean(s);

loglog(x,idc,'k:');
hold off;
legend('Measured','Simulated');
```

LIST OF REFERENCES

Cobbley, David A., "Multipoint LAN Conferencing," Digest of Papers, COMPCON Spring 1993, IEEE Computer Society Press, 1993, pp. 502-506.

Cornell, G., and Horstmann, C., "Core Java", SunSoft Press, Mountain View, CA, 1996.

Cinotti, M., Giordano, S., Romani, F., and Russo, F., "Implications of the Self-Similar Behaviour of Real Traffic Sources on the Local Queue Occupancy in Metropolitan Area Networks", Proceedings of IEEE Singapore International Conference on Networks/International Conference on Information Engineering 1995, 3-7 July 1995, pp. 431-435.

Dalgic, Ismail, Chien, William, and Tobagi, Foud A., "Evaluation of 10Base-T and 100Base-T Ethernets Carrying Video, Audio and Data Traffic," Proceedings IEEE INFOCOM 1994, IEEE Computer Society Press, 1994, pp. 1094-1102, vol. 3.

Flanagan, D., "Java in a Nutshell", O'Reilly and Associates, Inc, Sebastopol, CA, 1996.

Fowler, Henry J., and Leland, Will E., "Local Area Network Traffic Characteristics, with Implications Broadband Network Congestion Management", *IEEE Journal on Selected Areas of Communications*, no. 9, pp. 1139-1149, September 1991.

Harju, Jarmo, Kosonen, Ville-Pekka, and Li, Changhong, "Quality and Performance of a Desktop Video Conferencing System in the Network of Interconnected LANs," *Proceedings of the 19th Conference on Local Computer Networks*, IEEE Computer Society Press, October 2-5 1994, Minneapolis, MN, pp. 365-371.

Haykin, Simon S., "An Introduction To Analog and Digital Communications," John Wiley & Sons, Inc., US, 1989.

Leland, Will E., Taqqu, Murad S., Willinger, Walter, and Wilson, Daniel V., "On the Self-Similar Nature of Ethernet Traffic (Extended Version),"*IEEE /ACM Transactions on Networking*, vol. 2 no. 1, February 1994.

Leon-Garcia, A., "Probability and Random Processes for Electrical Engineers," Addison-Wesley Publishing Co., Reading, MA, 1994.

Likhanov, N., and Tsybakov, B., "Analysis of an ATM Buffer with Self-Similar ("Fractal") Input Traffic", Proceedings IEEE INFOCOM '95, pp. 985-992, vol. 3.

MathWorks, Inc., "MATLAB Reference Guide", MathWorks, Inc., Natick, MA, 1992.

Perloff, Micheal, "Videoconferencing for Command and Control," MILCOM 1993, IEEE, October 11-14 1993, Boston, MA, pp. 476-479, vol. 2.

Schwartz, M., "Broadband Integrated Networks", Prentice Hall, Upper Saddle River, NJ, 1996.

Stallings, William, "SNMP, SNMPv2 and RMON," Addison-Wesley Publishing Co., Inc., Reading, MA, 1996.

Stevens, R., "TCP/IP Illustrated, Volume 1", Addison-Wesley Publishing Co., Inc., Reading, MA, 1994.

Subramanian, S. N., and Le-Ngoc, T., "Traffic Modeling in a Multi-media Environment", 1995 Canadian Conference on Electrical and Computer Engineering, pp. 838-841, vol. 2.

Vetter, Ronald J., "Videoconferencing on the Internet," *Computer*, vol. 28 no. 1, pp.77-79, January, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

4. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

5. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

6. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

7. Prof. Hershel H. Loomis, Jr., Code EC/Lm 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

8. Prof. Murali Tummala, Code EC/Tu 3
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

9. Captain Randy Garcia 1
Marine Corps Tactial Systems Support Activity (MCTSSA)
PO Box 555171
Camp Pendleton, CA 92055-5171

10. Captain Lloyd Biggs 1
Marine Corps Tactial Systems Support Activity (MCTSSA)
PO Box 555171
Camp Pendleton, CA 92055-5171

11. Captain David Wells 1
Marine Corps Tactial Systems Support Activity (MCTSSA)
PO Box 555171
Camp Pendleton, CA 92055-5171

12. Captain Harlan Carvey 2
213 Sicily Rd.
Seaside, CA 93955